Processor Expert

CodeWarrior Plug-in for Freescale

User Manual

version 2.85

CONTENTS

1. Introduction

Both hardware and software design have progressed so much with the ever-advancing new technologies emerging everyday, but their interrelationships and interdependence have been mostly neglected. On one hand, we often see a good new hardware architecture but the software design is too expensive for such an architecture. On the other hand, the computerization of nearly all mechanical gadgets all over the modern world leads to the use of embedded computer systems. In situations where expense is a consideration, embedded computer systems with efficient software can significantly reduce the overall design cost.

Processor Expert Code Warrior plug-in is designed for **rapid application development** of embedded applications for a wide range of microcontrollers and microprocessor systems.

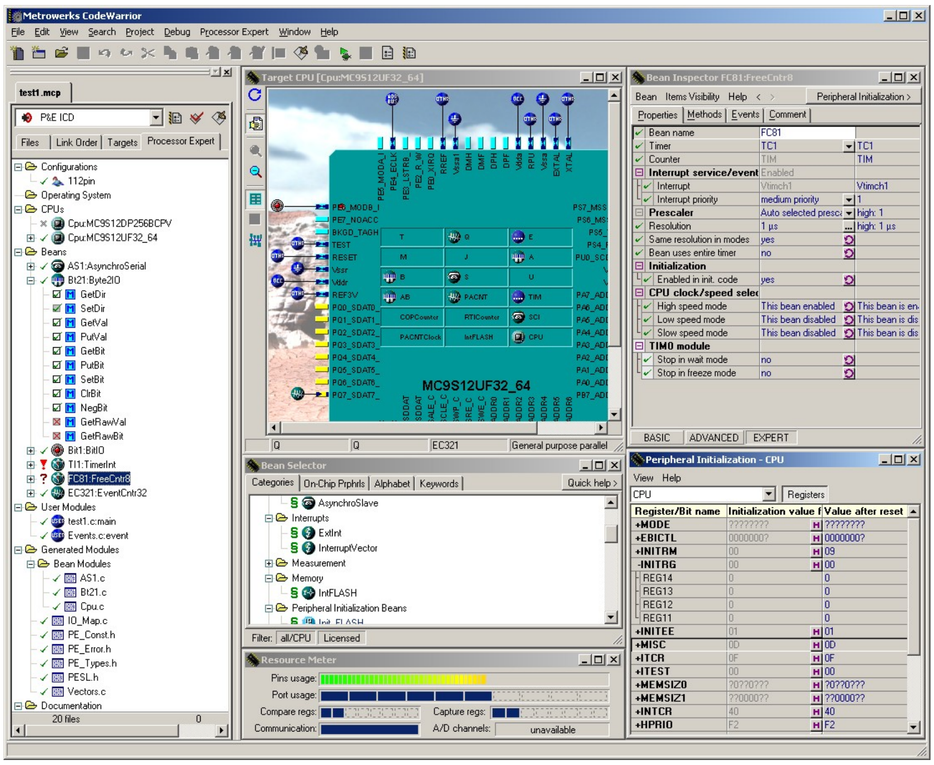
Processor Expert Main features

- The application is created from **components** called **Embedded Beans.**
- Embedded Beans encapsulate functionality of basic elements of embedded systems like CPU core, CPU on-chip peripherals, FPGA, standalone peripherals, virtual devices, and pure software algorithms and change these facilities to properties, methods and events (like objects in OOP).
- Processor Expert suggests, connects and generates the drivers for an embedded system hardware, peripherals or used algorithms. This allows the user to concentrate on the creative part of the whole design process.
- Processor Expert allows true **top-down** style of application design - the user starts the design directly by defining the application behavior instead of spending days just trying to make the chip work.
- Processor Expert works with an **extensible beans library** of supported microprocessors, peripherals and virtual devices.
- Processor Expert Peripheral Initialization beans generate effective initialization code for all on-chip devices and support all their features.
- Processor Expert allows to easily examine the details of the architecture and the relationship between the Embedded Bean setup and CPU control registers initialization.
- The user can create his/her own beans using the **Bean Wizard** external tool. See chapter *5 Bean Wizard Description* for details.

1.1. Processor Expert Plug-in Overview

The **Processor Expert** was originally developed as a stand-alone product. Now, to provide a more efficient and comfortable development environment, we integrated it as a plug-in to the CodeWarrior.

The CodeWarrior IDE menu contains a new menu item named **Processor Expert.** The Processor Expert plug-in generates code from the Embedded Beans and CodeWarrior manages the project files and the compilation and debugging processes.



Bean name	FC81	
Timer	TC1	TC1
Counter	TIM	TIM
Interrupt service/event	Enabled	
Interrupt	Vtimch1	Vtimch1
Interrupt priority	medium priority	1
Prescaler	Auto selected presca	high: 1
Resolution	1 µs	high: 1 µs
Same resolution in modes	yes	
Bean uses entire timer	no	
Initialization		
Enabled in init. code	yes	
CPU clock/speed selec		
High speed mode	This bean enabled	This bean is en
Low speed mode	This bean disabled	This bean is dis
Slow speed mode	This bean disabled	This bean is dis
TIM0 module		
Stop in wait mode	no	
Stop in freeze mode	no	

Register/Bit name	Initialization value f	Value after reset
+MODE	????????	????????
+EBICTL	0000000?	0000000?
+INITRM	00	09
-INITRG	00	00
REG14	0	0
REG13	0	0
REG12	0	0
REG11	0	0
+INITEE	01	01
+MISC	0D	0D
+ITCR	0F	0F
+ITEST	00	00
+MEMSIZ0	?0??0???	?0??0???
+MEMSIZ1	??0000??	??0000??
+INTCR	40	40
+HPRIO	F2	F2

Figure 1.1 - CodeWarrior IDE with Processor Expert plug-in active

How to create a new project

See the chapter 4 Processor Expert Tutorials or 3.1 Quick Start in Processor Expert for step-by-step instructions on how to start a new Processor Expert project.

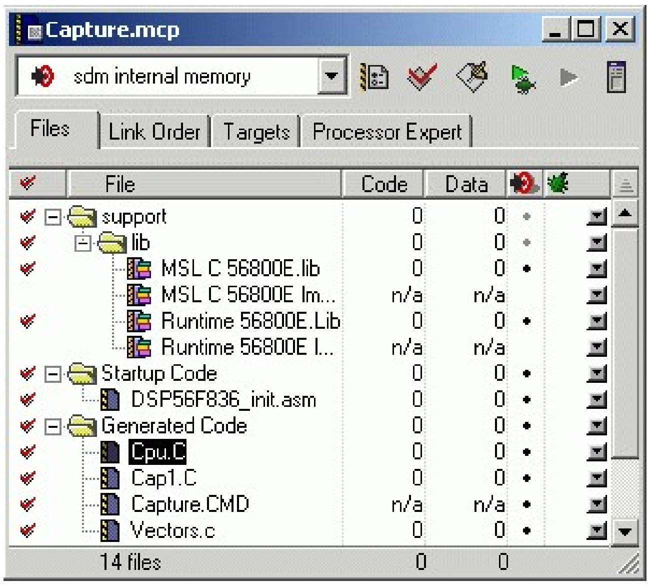
Compiler and Linker settings

To set the compiler and linker options, select the command **{TargetName} Settings** in the "Edit" menu in the Code Warrior main menu. You can find linker and compiler specific settings in the "Target" and "Linker" folders. The command **{TargetName} Settings** is not available when no project is open.

Where to find source code and user modules

Processor Expert generates all drivers during the code design process. The generated files are automatically inserted into the active (default) target in the CodeWarrior project. Generated files corresponding to Embedded Beans can be accessed in the "Generated Code" folder in the "Files" tab in the Code Warrior project window.

Other files, intended to be modified by users, are generated into the "User modules" folder in the "Files" tab in the Code Warrior Project window. A user can also add his/her own specific source code files into this folder. If the linker setting of the default target does not match the CPU in the Processor Expert project, the user is asked whether to automatically set the correct linker settings in the default target or to create a new target with correct linker settings. In the latter case the files will be generated in the new target (more information about the CodeWarrior Project panel can be found in the CodeWarrior documentation). For more information on generated files please see the chapter 3.5.1 Code Generation.



Capture.mcp

sdm internal memory

Files | Link Order | Targets | Processor Expert

	File	Code	Data		
✓	support	0	0	•	
✓	lib	0	0	•	
✓	MSL C 56800E.lib	0	0	•	
	MSL C 56800E Im...	n/a	n/a		
✓	Runtime 56800E.Lib	0	0	•	
	Runtime 56800E I...	n/a	n/a		
✓	Startup Code	0	0	•	
✓	DSP56F836_init.asm	0	0	•	
✓	Generated Code	0	0	•	
✓	Cpu.C	0	0	•	
✓	Cap1.C	0	0	•	
✓	Capture.CMD	n/a	n/a	•	
✓	Vectors.c	0	0	•	
	14 files	0	0		

Figure 1.2 - CodeWarrior project panel

1.2. Benefits of Embedded Beans and Processor Expert Technology

The key benefit of **Embedded Beans** is the same as using components in software design environments like Microsoft Visual Basic or Borland Delphi. In comparison with components used within these products, Embedded Beans provide hardware encapsulation in the form of a platform-independent standard. Different players in the embedded market should benefit from such a standardization approach.

Microprocessor producers

Each year microprocessor producers introduce many new microprocessor families or derivatives. As the complexity of microprocessors increases, programmers must handle more and more registers to get the required functionality. Classical development tools usually do not support the rapid prototyping phase of design, and classical programming languages are not able to efficiently describe the on-chip peripherals structure. On the other hand, microprocessor producers need to speed up the learning, design and coding processes for their customers.

For the designer, Processor Expert and its configuration and code generation features completely eliminate the necessity to be otherwise preoccupied with hardware dependencies. Processor Expert could even suggest the right member of a microprocessor family for the specific application.

Producers of intelligent peripheral I/Os and other devices

Complex and feature-rich peripherals and controllers require a big effort to use them efficiently, even if device drivers are supplied by the factory. But imagine the possibility of supporting customers with components providing a standard software interface that allows building applications and easily uses new hardware device features.

And yes, the Processor Expert environment allows this - customers can easily download new components from the internet and install them into Processor Expert.

Producers of hardware of microprocessor systems

Microprocessor boards that are to be programmed by a customer must be well supported by software. Processor Expert can handle software configuration and generation of drivers for microprocessor devices and off-chip peripheral devices. Creating an application using Processor Expert takes usually 70% less time than with standard Integrated Development Environments (IDEs) containing only a source code editor/compiler/debugger.

Producers of compilers, hardware emulators and simulators

Processor Expert is able to cooperate with other tools and IDEs because it works on a higher level of abstraction. Tools suppliers can increase the attractiveness of their tools with Processor Expert features.

Producers of programmable logic (FPGA,..)

When a customer designs his own FPGA-based peripheral, it is possible to "bean-it" - to include its standard form into the Processor Expert Embedded Beans palette. Then the design could be reused in software and supplied to the customers of the FPGA designer with a full software support.

Producers of OS

Processor Expert can be used to build an OS kernel or OS drivers. Also, thanks to Processor Expert open component architecture and support of pure software beans, Processor Expert can be used to build applications benefiting from underlying operating system services.

Educational institutes

Microprocessor-oriented courses can benefit from the concentration of knowledge of microprocessor structures and hardware independence delivered by Processor Expert. Design of applications starts from a definition of functionality, which is then obtained very quickly by building the application from Embedded Beans. Students can get the results very fast without struggling with problems that are not related to the subject of the course (e.g., compiler bugs, errors in the documentation and so on).

Hardware and software developers

Shortening of the design and learning phase, speeding up the deployment of new components, full use of hardware using tested software components, reducing time and cost of design - all these are keys to success provided by Processor Expert.

1.3. Features

Processor Expert has built-in knowledge (internal definitions) of the entire CPU with all of its units and integrated peripherals. The CPU units and peripherals are encapsulated into configurable components called Embedded Beans, each of which provides a set of useful properties, methods and events.

An intuitive and powerful User Interface (UI) allows the user to define the system behavior in several steps. A simple system can be created just by selecting the necessary beans, setting their properties to the required values and maybe also dragging and dropping some of their methods to the user part of the project source code.

PE key components

- Graphical IDE
- Built-in detailed design specifications of Freescale devices
- Code generator

PE key features

- Design-time verifications
- CPU selection from multiple CPU derivatives available
- CPU pin detailed description and structure viewing
- Configuration of functions and settings for the selected CPU and its peripherals
- Definition of system behavior during initialization and at runtime
- Design of application from pre-built functional components (called BEANS)
- Design of application using bean methods (user callable functions) and events (templates for user written code to process events, e.g. interrupts)
- Customization of beans and definition of new beans
- Tested drivers
- Library of components/beans for typical functions (including virtual SW beans)
- Verified reusable beans allowing inheritance
- Verification of resource and timing contentions
- CPU resource meter/balancing
- Concept of project panel with ability to switch/port between CPU family derivatives
- Code generation for components included in the project
- Implementation of user written code
- Interface with Freescale CodeWarrior

PE based tool solution offers the following advantages to Freescale CPU customers:

- In all phases of development, customers will experience substantial reductions in
  - development cost
  - development time
- Additional benefits in product development process are
  - Integrated development environment increases productivity
  - Minimized time to learn Freescale CPU

- Rapid prototyping of entire applications
- Modular and reusable functions
- Easy to modify and port implementations

Integrated development environment increases users' productivity

- "This tool lets me produce system prototypes faster because the basic setup of the controller is easier. This could mean that I will implement more of my ideas into a prototype application having a positive effect on the specification-, analysis- and design-phase. PE justifies its existence even when used for this purpose alone!"
- "This system frees you up from the hardware considerations and allows you to concentrate on software issues and resolve them thoroughly."
- "Very good for CPUs with embedded peripherals. It significantly reduces project development time."

The following are the primary reasons why users feel that way:

- PE has built-in knowledge (internal definition) of the entire CPU with all its integrated peripherals.
- PE encapsulates functional capabilities of CPU elements into concepts of configurable beans.
- PE provides an intuitive graphical UI, displays the CPU structure, and allows the user to take advantage of predefined and already verified beans supporting all typically used functions of the CPU.
- Applications are designed by defining the desired behavior using the component settings, drag & drop selections, utilizing the generated methods and events subroutines, and combining the generated code with user code.
- PE verifies the design based on actual CPU resource and timing contentions.
- PE allows the efficient use of the CPU and its peripherals and building of portable solutions on a highly productive development platform.

Minimized time to learn the CPU

There are exciting possibilities in starting a new project if the user is starting from ground zero even if the user is using a new and unfamiliar processor.

- The user is able to utilize the CPU immediately without studying the CPU's documentation.
- The user is able to implement simple applications even without deep knowledge of programming.
- PE presents all necessary information to the user using built-in descriptions and hints.
- PE has built-in tutorials and example projects.

Rapid prototyping of entire applications

"Processor Expert allows the users to try several different approaches in real time, picking and choosing the best of each for the final solution. Users are not confined to a pre-determined linear track to a solution."

- Easy Build of application - based on system functional decomposition (top-down approach)
- Easy/Auto CPU selection
- Easy/Auto CPU initialization
- Easy/Auto initialization of each internal peripheral
- Simple development of reusable drivers
- Simple implementation of interrupt handlers

- Inherited Modularity and reuse
- Inherited ease of implementation of system hardware and software/firmware modifications

Modular and reusable functions

Processor Expert greatly decreases the start-up time and minimizes the problems of device idiosyncrasies.

- It uses the concept of a function encapsulating entity (called Embedded Bean) with supporting methods and events
- Uses a library of predefined beans
- Uses a concept of device drivers and interrupt handlers that are easy to reapply
- Uses a concept of well documented programming modules to keep the code well organized and easy to understand

Easy to modify and port implementations

PE allows optimal porting to a previously unused processor.

- Supports multiple devices within a project and makes it extremely easy to switch them
- Supports desired changes in the behavior of the application with an instant rebuild
- Supports interfacing of CodeWarrior

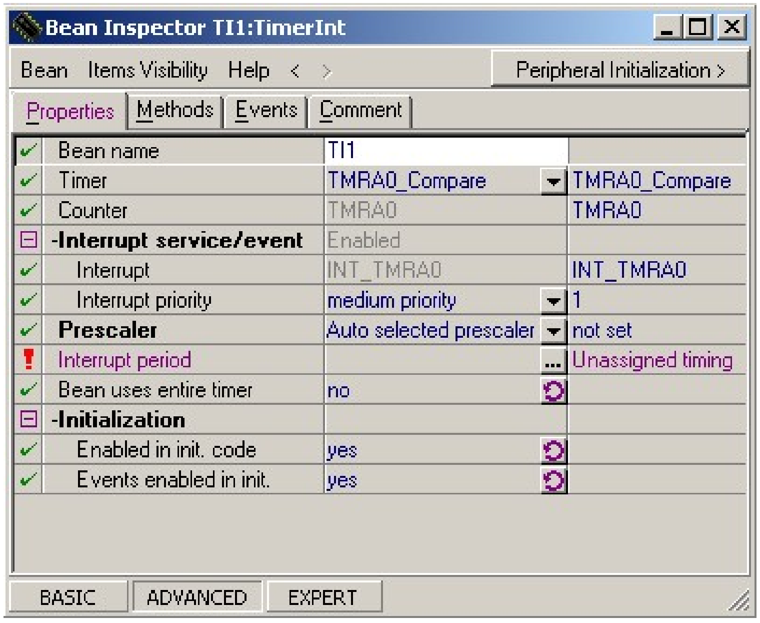
1.4. Concepts

The main task of **Processor Expert** is to manage CPU and other hardware resources and to allow virtual prototyping and design.

Code generation from beans, the ability to maintain user and generated code, and an event based structure significantly reduce the programming effort in comparison with classic tools.

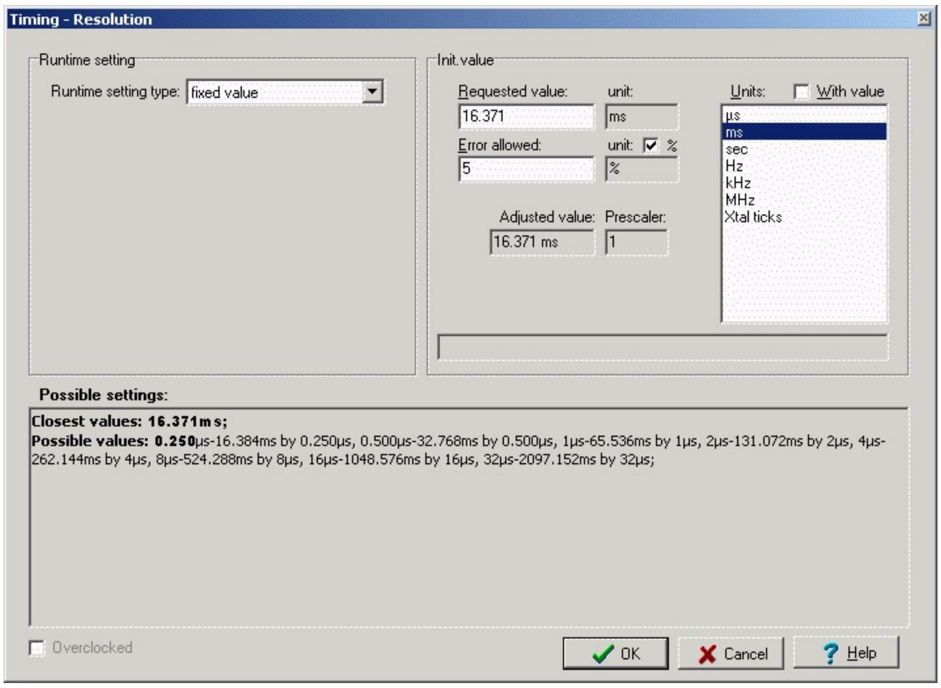
Embedded Beans

Component (bean) is the essential encapsulation of functionality. For instance the TimerInt bean encapsulates all CPU resources that provide timing and hardware interrupts on the CPU.



✓	Bean name	TI1	
✓	Timer	TMRA0_Compare	TMRA0_Compare
✓	Counter	TMRA0	TMRA0
⊟	**-Interrupt service/event**	Enabled	
✓	Interrupt	INT_TMRA0	INT_TMRA0
✓	Interrupt priority	medium priority	1
✓	**Prescaler**	Auto selected prescaler	not set
!	Interrupt period		Unassigned timing
✓	Bean uses entire timer	no	
⊟	**-Initialization**		
✓	Enabled in init. code	yes	
✓	Events enabled in init.	yes	

Figure 1.3 - Example of TimerInt bean (periodical event timer) properties



Runtime setting type: fixed value

Requested value: 16.371 unit: ms

Error allowed: 5 unit: ✓ % %

Adjusted value: 16.371 ms Prescaler: 1

Possible settings:

Closest values: 16.371ms;
Possible values: 0.250µs-16.384ms by 0.250µs, 0.500µs-32.768ms by 0.500µs, 1µs-65.536ms by 1µs, 2µs-131.072ms by 2µs, 4µs-262.144ms by 4µs, 8µs-524.288ms by 8µs, 16µs-1048.576ms by 16µs, 32µs-2097.152ms by 32µs;

Figure 1.4 - Timing dialog allows a user friendly setting of beans' timing

You'll find many components that we call Embedded Beans in the Processor Expert Bean selector window. These components were selected to cover the most commonly required functionality used for microcontroller applications - from handling port bit operations, external interrupts, and timer modes up to serial asynchronous/synchronous communications, A/D converter, I2C, CAN, etc.

A bean provides a **clear interface**. By setting **properties**, a user defines the future behavior of the bean in runtime. The user controls properties in design time by using the Beans Inspector. Runtime control of the bean function is done by the **Methods**. **Events** are interfacing hardware or software events invoked by the bean to the user's code.

The user can enable or disable the appearance (and availability) of methods of the bean in generated source code. Disabling unused methods could make the generated code shorter. See chapter *3.6.1 General Optimizations* for details.

Events, if used, can be raised by interrupt from the hardware resource (timer, SIO,..) or by pure software reason (overflow,..) in application runtime. You can enable or disable interrupts using bean **methods** and define priority for event occurrence and for executing its Interrupt Service Routine (ISR). The hardware ISR provided by the bean handles the reason for the interrupt. If the interrupt vector is shared by two (or more) resources, then this ISR provides the resource identification. Then the user is notified by calling the **user event handling code.**

Creating Applications

Creation of an application with Processor Expert on any microcontroller is very fast. First choose and set up a CPU bean, add other beans, modify their properties, define events and select Generate Code. Processor Expert generates all code (well commented) from beans according to your settings. See chapter *3.5.1 Code Generation* for details.

This is, of course a only part of the application code that was created by the "virtual application engineer" - Processor Expert CPU knowledge system and solution bank. The solution bank is created from hand written and tested code optimized for efficiency. These solutions are selected and configured in the code generation process.

Enter your code for desired events, provide main code, add existing source code - and build the application using classic tools - compiler, assembler - and debug it before the final burn-in. These are typical steps when working with Processor Expert.

Other beans help you to very quickly include pictures, files, sounds, and string lists in your application .

The other beans can be obtained from www.processorexpert.com or created from existing sources, for instance FFT. Other beans can incorporate already existing beans. They can inherit their properties, methods, and events.

Imagine that you want to share a bean with other developers. For example a bean that can drive an LED segment display. Because it is used often for different hardware configurations - on different CPU pins - then it must be portable and independent of CPU resources.
A lot of tasks and algorithms can be incorporated into a bean. Such beans are called software (SW) beans. SW beans can be pure SW beans (FFT) or can inherit even multiple beans that encapsulate HW resources. The advantage is independence on a physical layer, portability and sharing of once written and tested code.

For our example we simply choose as parents BitIO, BitsIO or ByteIO and TimerInt beans from the bean library. The new LED display bean will provide the properties of a bean reference type for this bean. In design time this allows the new bean access to its parents' properties and defines the physical connection pins or timer resources. Additionally, the new bean will have its own properties and methods. Methods and events can be constructed using the parent bean's methods.

Don't be concerned about the complexity of this process - simply choose from the Processor Expert Tools menu the Beans Wizard tool which makes all the arrangements for you. You only need to enter the code of methods and events, save new bean and install it on the Beans Palette or share it with others.

For additional information about Processor Expert and beans libraries please go to online help or the www.processorexpert.com website.

Processor Expert has built-in knowledge (internal definitions) of the entire CPU with all of its units and integrated peripherals. The CPU units and peripherals are encapsulated into configurable components called Embedded Beans and the configuration is fast and easy using a graphical Bean Inspector.

Peripheral Initialization Beans (a subset of Embedded Beans) allows the user to **setup initialization of the particular on-chip device** to any possible mode of operation. The user can easily view all initialization values of the CPU produced by Processor Expert with highlighted differences between the last and current properties settings.

Processor Experts performs a **design time checking of the settings** of all beans and reports errors and warnings noticing user about wrong property values or collisions in the settings with other beans in the project.

Processor Expert contains many **useful tools for exploring a structure of the target CPU** showing the details about the allocated on-chip peripherals and pins.

Processor Expert generates a **ready-to-use source code** initializing all on-chip peripherals used by the bean according to the bean setup.

1.5. Terms and Definitions Used in Processor Expert

Bean - An Embedded Bean is a component that can be used in Processor Expert. Embedded Beans encapsulate the functionality of basic elements of embedded systems like CPU core, CPU on-chip peripherals, standalone peripherals, virtual devices and pure software algorithms and wrap these facilities to properties, methods, and events (like objects in OOP). Beans can support several languages (ASM, ANSI C, Modula and others) and the code is generated for the selected language.

Bean Inspector - window with all parameters of a selected bean: properties, methods, events.

CPU Bean - bean which encapsulates the CPU core initialization and control. This bean also holds a group of settings related to the compilation and linking (Stack size, Memory mapping, linker settings etc..). Only one CPU bean can be set active as the target CPU. See chapter *3.2.2 CPU Beans* for details.

Bean Driver - Bean drivers are the core of Processor Expert code generation process. Processor Expert uses drivers to generate the source code modules for driving an internal or external peripheral according to the bean settings. A Bean can use one or more drivers.

Events - are used for processing events related to the bean's function (errors, interrupts, buffer overflow etc.) by user-written code. See chapter *3.2.1 Embedded Beans* for details.

External user module - external source code attached to the PE project. The external user module may consist of two files: implementation and interface (*.C and *.H).

Internal peripherals - internal devices of the CPU (ports, timers, A/D converters, etc. usually controlled by the CPU core using special registers).

ISR - Interrupt Service Routine - code which is called when an interrupt occurs.

Methods - user callable functions or sub-routines. The user can select which of them will be generated and which not. Selected methods will be generated during the code generation process into the bean modules.

Module - source code module. Could be generated by Processor Expert (Bean modules, CPU Module, events.c) or created by the user and included in the project (user module).

OOP - Object-oriented programming (OOP) was invented to solve certain problems of modularity and reusability that occur when traditional programming languages such as C are used to write applications.

PE - Abbreviation of Processor Expert which is often used within this documentation.

PESL (Processor Expert System Library) is dedicated to power programmers, who are familiar with CPU architecture - each bit and each register. PESL provides the macros to access the peripherals directly, so PESL should be used only in some special cases. See chapter *3.7.1 Processor Expert System Library* for details.

Peripheral Initialization bean - encapsulates the whole initialization of the appropriate peripheral. Beans that have the lowest levels of abstraction and usage comfort. See chapter *3.2.1.1 Bean Categories* for details. They usually do not support any methods or events except the initialization method. The rest of the device driver code needs to be written by hand using either PESL or direct control of the peripheral registers. See chapter *3.7 Low-level Access to Peripherals* for details.

Popup menu - this menu is displayed when the right mouse button is pressed on some graphical object.

Properties - parameters of the bean. Property settings define which internal peripherals will be used by the bean and also initialization and behavior of the bean at runtime.

Target CPU - the CPU derivative used in a given project.

Template - Bean Template is a bean with preset parameters.

User-defined Bean Template - User-defined bean template is a bean with preset parameters saved under a selected name. Also the name of the author and short description can be added to the template.

User module - a source code module created or modified by the user. (Main module, event module or external user module).

2. User Interface

Menu

Processor Expert menu is integrated in the CodeWarrior IDE menu. It contains a new item named **"Processor Expert"**.
See Processor Expert plug-in Main menu page for description of individual items.

The user interface of Processor Expert consists of the following windows (integrated in CodeWarrior IDE):

Project Editing Windows

- Project panel with beans (including CPU(s)), external modules and documentation included in project. Project Panel supports several configurations of one project.
- Inspector - a window which allows the user to setup Beans and Configurations of the project.
- Bean Selector - shows all supported beans in the appropriate version of the Processor Expert including CPU beans and bean templates.
- Target CPU - a window graphically showing CPU package, structure and beans connected to internal peripherals. Allows to easily add beans related to a specific peripheral to the project using a pop-up menu of the peripheral.

Project Information Windows

- Error window - a window with errors, warning messages and hints from project checking, generation and from external tools
- CPU Timing Model - a window showing the target CPU's timing.
- Peripheral Initialization - shows overview of peripheral initialization settings for the current CPU.
- Peripherals Usage Inspector - a window showing which bean allocates which on-chip peripheral.
- Resource Meter - a window displaying the amount of the target CPU's resources already allocated.
- Memory Map - a window showing the CPU address space and internal and external memory mapping.

Processor Expert Overview Windows

- Installed Beans Overview - this window contains information about installed beans in the current version of Processor Expert.
- CPU Types Overview - a window displaying the list of the database's CPUs supported by the current version of Processor Expert.
- CPU Parameters Overview - a window providing access to the CPU's database.
- PDF Search - a window allows the user to quickly browse in a PDF documentation for the CPU.

Dialogs

There are the following dialogs for setting the Processor Expert environment:

- Environment Options - Processor Expert plug-in environment options
- Tools Setup - setup dialog box for the tools and the tools menu

There are the following dialog for setting the Processor Expert project:

- Project Options - project options are options concerning the current project and options for the current CPU.

2.1. Main Menu

The Processor Expert Plug-in is integrated into the CodeWarrior IDE application. The CodeWarrior IDE main menu contains a new menu item named "Processor Expert".

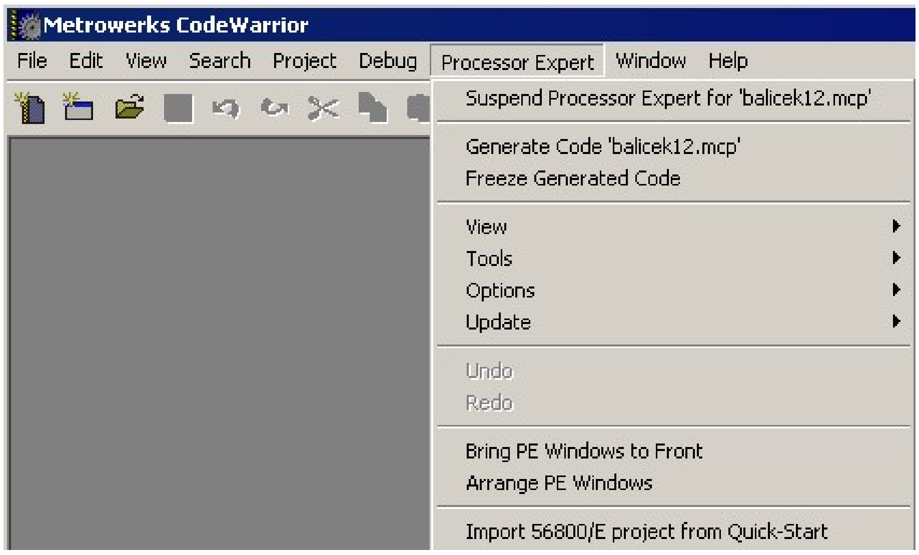


Figure 2.1 - Processor Expert's Integrated Menu

The Processor Expert's plug-in menu consists of the following items:

- **Suspend/Open Processor Expert**
- **Generate Code "ProjectName.mcp"**
- **Freeze Generated Code**
- **View**
- **Tools**
- **Options**
- **Update**
- **Undo**
- **Redo**
- **Bring PE Windows to Front**
- **Arrange PE Windows**

- **Import 56800/E project from Quick-Start**

Note: Processor expert's help is placed in the Codewarrior's 'Help' menu.

Suspend/Open Processor Expert - Suspend Processor Expert will disable usage of Processor Expert for the currently opened project. Open Processor Expert will enable the Processor Expert for the current project.

When PE is enabled for a project that had never had PE activated, PE will popup a dialog that notifies the user that PE will add files (IOMap.c/h) containing declarations for the peripheral modules to the project, so the user should know how to adapt the project himself (there might be other files as well (e.g. linker parameter file) that can cause conflicts too). This command is intended for **experienced users only.** New Processor Expert project should be created using the **File | New** command.

Code Generation - Invokes code generation for the current project. The generated files are automatically inserted into the active (default) target in the CodeWarrior's project. Generated files corresponding to the Embedded Beans can be accessed in the "Generated Code" folder in the "Files" tab in the CodeWarrior project window. The other files, intended to be modified by the user, are generated into the "User modules" folder in the "Files" tab in the CodeWarrior project window. A user can also add specific source code files into this folder. If the linker setting of the default target does not match the CPU in the Processor Expert project, the user is asked whether to automatically correct linker settings in the default target or to create a new target with correct linker settings. In the latter case the files will be generated in the new target (see also chapters 3.5.1 Code Generation and 3.5.1.1 Linker Dialog).

Freeze Generated Code - This option will freeze the state of the generated code and the code generation will be disabled until the user will un-check this option. All beans and project settings will became read-only and it won't be possible to add or remove any beans. Processor Expert won't make any changes to the source code. Processor expert, if it detects any changes since last code generation, will offer code generation before switching to the 'frozen' mode.

View

- **Project Panel** - displays the Processor Expert plug-in Project panel.
- **Inspector** - displays inspector window for the currently selected item of the project (Bean, CPU bean, Peripheral Initialization bean, Configuration)
- **Bean Selector** - shows the Bean Selector. Bean Selector shows all supported beans in the appropriate version of the Processor Expert plug-in including CPU beans.
- **Target CPU Package** - displays the Target CPU Window in CPU Package view. This window displays the target CPU (CPU selected as destination) with its peripherals and pins.
- **Target CPU Block Diagram** - displays the Target CPU Window in Block Diagram view. This window displays the target CPU (CPU selected as destination) block diagram with its peripherals.
- **Error Window** - displays the Error window. This window displays errors, warnings, and hints.
- **Target CPUTiming Model** - displays the CPU timing hierarchy.
- **Peripheral Initialization** - opens the Peripheral Initialization window for the Target CPU. (this command is available only if a target CPU is selected)
- **Peripherals Usage** - shows the CPU Peripherals Usage window.
- **Resource Meter** - displays the Resource Meter window. The Resource Meter shows the current status of a chip resources usage (or availability).
- **Memory Map** - opens the Memory Map window. This window shows the CPU address space and internal and external memory mapping.
- **Installed Beans Overview** - displays a list of installed beans and CPUs with additional information about

bean drivers and projects with typical settings.

- **CPU Types Overview** - displays the CPU overview window with a tree of supported CPUs (the active CPU's package is displayed in the Target CPU Window).
- **CPU Parameters Overview** - displays the CPU parameters overview table and the query dialog that provides help for the selection of the most adequate processor.

Tools

- **Tool #1**
- **Tool #2....** - optionally, any other external tool can be added. The tools can be added, modified or deleted in the "Tools Setup" dialog.

Note: Bean Wizard can be added also to the Tools menu. Help for the Bean Wizard can be found in the Bean Wizard.

Options

- **Environment Options**, **Project Options**, **Application Options** - opens an appropriate page within the Processor Expert Options dialog window that allows to customize all settings related to the environment and project. See chapter *2.1.1 Processor Expert Options* for details.
- **Tools Setup** - allows external tools to be included in the Processor Expert's plug-in environment. The tools may then be accessed via the "Tools" menu. The setting changes will take effect after the restart of the CodeWarrior application. See chapter *2.1.2 Tools Setup* for details.
- **Save desktop** - saves the desktop settings (windows' position) to the .DSK file. The desktop file can also be saved automatically if the option **Environment Options | Autosave desktop** is enabled in Environment Options.

Update

- **Update Processor Expert from Package** - this command allows the user to update or add new beans from compressed packages (*.PEupd) that can be downloaded from the Processor Expert web site. It is possible to select more beans in the selected directory using a multi-select function. To add more beans select the requested bean using the mouse and holding down the CTRL (or Shift) key.

  The **information on the package content** is shown when the package is selected within an opening dialog window.

- **Check Processor Expert Web for updates** - check for updates on the Processor Expert web site. If there is any news for your version, Processor Expert offers you to open the corresponding page in a default Internet browser.

Undo [actionname]

Restores the state of the project before a last operation. **This command affects only changes in the project** (i.e. adding or removing beans, disabling beans etc.). It doesn't work on the source code editor actions. Functionality of this command is influenced by the option **Environment Options | Number of UNDO operations**. The '0' value of this option will disable the functionality of this command.

Redo [actionname]

Applies again the change previously discarded by a use of the Undo command. **This command affects only changes in the project** (i.e. adding or removing beans, disabling beans etc.). It doesn't work on the source code editor actions. Functionality of this command is influenced by the option **Environment Options | Number of UNDO operations**. The '0' value of this option will disable the functionality of this command.

Bring PE Windows to Front

Sets the main Processor Expert's windows to the front on the screen.

Arrange PE Windows

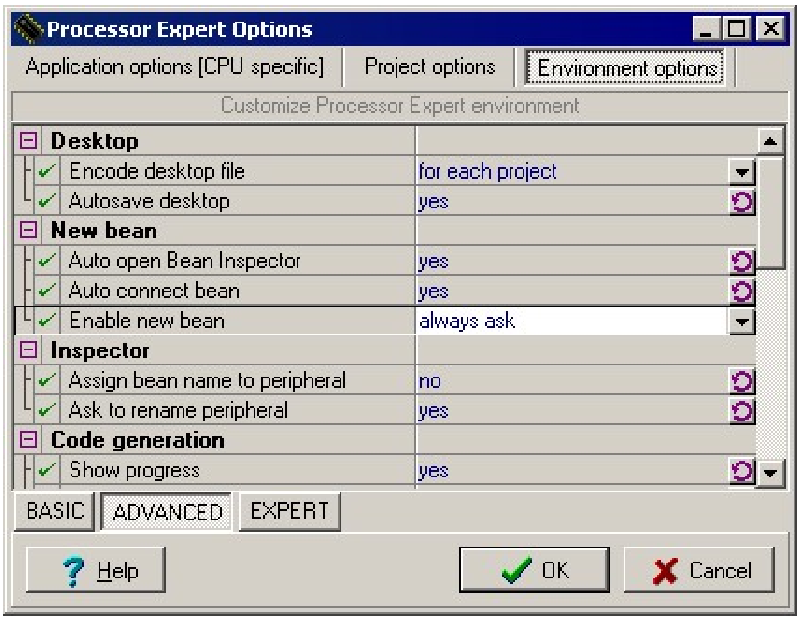
Arranges all open windows to the default placement on the screen. (Project Panel, Bean Selector, Cpu Panel, Error Window, Resource Meter, Bean Inspector)

Import 56800/E project from Quick-Start

Imports an old Quick-Start project for 56800/E derivatives and creates the appropriate Processor Expert project with corresponding CPU bean and Peripheral Initialization beans. For more details please refer to the chapter 3.3.9 Import 56800/E Project From Quick-Start.

2.1.1. Processor Expert Options

Processor Expert | Options | Environment Options
Processor Expert | Options | Project Options
Processor Expert | Options | Application Options
Processor Expert options allows to customize all Processor Expert's settings within one dialog window.



Desktop	
Encode desktop file	for each project
Autosave desktop	yes
New bean	
Auto open Bean Inspector	yes
Auto connect bean	yes
Enable new bean	always ask
Inspector	
Assign bean name to peripheral	no
Ask to rename peripheral	yes
Code generation	
Show progress	yes

Figure 2.2 - Environment Options Example

The options are organized within three pages

- **Application Options** - options for the Processor Expert code generation (for one target CPU). They are local, i.e they are valid only for the current application (an application is the subset of the project that concerns a given target processor).

- **Environment Options** - options related to Processor Expert's environment behavior.
- **Project Options** - options concerning the current project (all target CPUs).

The item description for an item is provided as a hint when the user places mouse cursor on the item. Press **Help** button to open the options description pages.

Basic, Advanced and **Expert** buttons allow to customize the amount of options shown along to the user's experience level.

This window uses a limited version of Processor Expert Inspector to show the options information. Thus the way of changing options is very similar to the way of configuring a bean or configuration. See chapter *2.5.1 Inspector Items* for details.

2.1.2. Tools Setup

Processor Expert | Tools | Tools Setup

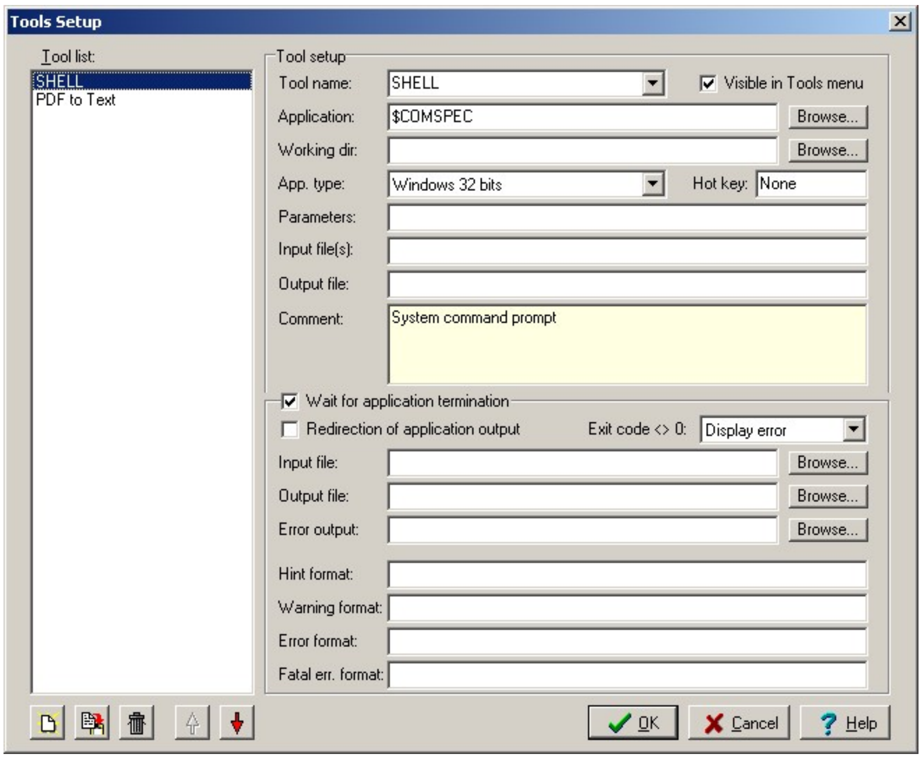
Tools Setup - allows to include other tools in the Processor Expert environment. The tools may then be accessed via the **Tools** menu.

Options

Notice: *Most of the options allow to use macros allowing an access to various Processor Expert and system values. See chapter 2.1.2.1 Tools Setup Macros* for details.
The following options are available for every tool :

- **Tool name** - name of the tool, as it appears in the Tools menu.
- **Visible in Tools menu** - whether the tool is available in the Tool menu (it may not be necessary to let it appear in the menu if the settings are meant only for internal make)
- **Application** - the full name (name and path name) of the application (executable file, EXE or COM extension).
- **Working dir** - working directory of the application
- **Application type**
  - *MS-DOS real* - MS-DOS real mode application.
  - *MS-DOS protected* - MS-DOS protected mode application.
  - *Windows 16-bit* - 16-bit MS Windows application.
  - *Windows 32-bit* - 32-bit MS Windows application.
  - *Autodetect* - auto detection (enabled under the Windows NT, Windows 2000 and newer).
- **Hot Key** - Hot Key for launching the tool.
- **Parameters** - parameters of the application.
- **Input file(s)** - only for backward compatibility, value of the $OUTPTH macro (see macros).
- **Output file(s)** - only for backward compatibility, value of the $IN?PTH macro (See chapter *2.1.2.1 Tools Setup Macros* for details.).
  There is no warranty that these items will be supported in the next version of Processor Expert.
- **Comment** - any text describing the tool.

- **Wait for application termination** - Processor Expert waits for application termination before executing any other operation (it has the advantage of reserving the error window for the application).
  - **Redirection of application output** - Processor Expert captures the standard output of the application and displays errors in the Message Window.
    - **Input file** - name of the input file for the application. If you don't specify any, a temporary file will be created.
    - **Output file** - name of the file for the redirection of the application output. If you don't specify any, a temporary file will be created.
    - **Error output** - name of the file for the redirection of the application error output. If you don't specify any, a temporary file will be created.
    - **Hint format** - format of the hint messages.
    - **Warning format** - format of the warning messages.
    - **Error format** - format of the error messages.
    - **Fatal error format** - format of the fatal error messages.
  - **Exitcode <> 0** - defines the action that will be done when the exit code of the application will not be zero. The following actions are possible:
    - **Ignore** - no action
    - **Display error** - an error message will be displayed
    - **Show output file** - opens the file defined as a tool output file in the editor.



Tool name: SHELL
Visible in Tools menu: ✓
Application: $COMSPEC
Working dir:
App. type: Windows 32 bits
Hot key: None
Parameters:
Input file(s):
Output file:
Comment: System command prompt
Wait for application termination: ✓
Redirection of application output: ☐
Exit code <> 0: Display error
Input file:
Output file:
Error output:
Hint format:
Warning format:
Error format:
Fatal err. format:

Figure 2.3 - Tools Setup Dialog

2.1.2.1. Tools Setup Macros

Global macros

Global macros can be used in the following tools setup options: *Application, Working directory, Parameters, Input file(s), Output file, Input files for redirection, Output file for redirection, Error output file for redirection, Format of all messages. See chapter 2.1.2 Tools Setup* for details.

- **$PRJNAME** - current name of the project without extension
- **$DIRPRJ** - current directory of the project, absolute path, terminated with a backslash
- **$DIRDRV** - current destination directory of drivers, absolute path, terminated with a backslash
- **$DIRRELDRV** - current destination directory of drivers, relative or absolute path from Project Options setting, terminated with a backslash ($DIRRELEVENT+$EVENTTODRV)
- **$DIREVENT** - current destination directory of main and event modules, absolute path terminated with a backslash
- **$DIRRELEVENT** - current destination directory of main and event modules, relative or absolute path from Project Options setting
- **$DIRBIN** - current destination directory of binary files (maker, linker and object), absolute path terminated with a backslash
- **$DIRRELBIN** - current destination directory of binary files, absolute or relative path starting from Project Options setting, terminated with a backslash ($DIRRELEVENT+$EVENTTOBIN)
- **$DRVTOEVENT** - relative path from drivers directory to main and event modules directory (drivers - Driver subdir., event - Main and event dir. from Project Options)
- **$EVENTTODRV** - relative path from main and event modules directory to drivers directory
- **$EVENTTOBIN** - relative path from main and event modules directory to binary files directory
- **$DIR_PE** - system directory of Processor Expert, absolute path, terminated with a backslash
- **$FILEDIR** - directory of the file that is currently edited or directory of the file that will be opened in External Text Editor, terminated with a backslash.
- **$FILENAME** - name of the file that is currently edited or name of the file that will be opened in External Text Editor (including extension).
- **$FILENAM** - name of the file that is currently edited or name of the file that will be opened in External Text Editor (without extension).
- **$GOTOLINE** - can be used only with the External Text Editor. It is replaced by the line number.
- **$?FILE(Question)** - name of the file set manually, Question is displayed to the title of window
- **$?PARAM(Question, Default)** - parameters set manually, Question is displayed to the title of window, Default is a default value
- **$COMSPEC** - setting of the COMSPEC variable in the Windows environment
- **$MAKEFILE** - name of the currently used makefile for the current project.

Global macros after code generation

The following macros are supported only after successful code generation. *Can be used in the same items as the Global Macros.*

- **$GENDIRPRJ** - directory of the project during last successful code generation, absolute path, terminated with a backslash
- **$GENDIRDRV** - destination directory of drivers during last successful code generation, absolute path, terminated with a backslash
- **$GENDIRRELDRV** - same as $GENDIRDRV during last successful code generation, only relative path ($GENDIRRELEVENT+$GENEVENTTODRV)
- **$GENDIREVENT** - destination directory of main and event modules during last successful code generation, absolute path terminated with a backslash
- **$GENDIRRELEVENT** - same as $DIRRELEVENT during last successful code generation
- **$GENDIRBIN** - destination directory of binary files during last successful code generation
- **$GENDIRRELBIN** - same as $GENDIRBIN during last successful code generation, only relative path ($GENDIRRELEVENT+$EVENTTOBIN)
- **$GENDRVTOEVENT** - relative path from drivers directory to main and event modules directory during last code generation
- **$GENEVENTTODRV** - relative path from main and event modules directory to drivers directory during last code generation
- **$GENEVENTTOBIN** - relative path from main and event modules directory to binary files directory during last code generation
- **$GENPRJNAME** - name of the project during last successful code generation

Macros in format

List of macros that can be used for definition of tool output messages format:

- **$ERRPTH** - name of file where errors were found
- **$ERRMSG** - full / partial error message
- **$FIRROW** - first row position
- **$FIRCOL** - first column position
- **$LASROW** - last row position
- **$LASCOL** - last column position
- **$MSGSKP** - skip next string
- **$MSGEND** - end/continuation of error message
- **$MSGSTR**"string1"string2 - string1 is written to error message and then string2 is skipped as in command $MSGSKP

Backward compatibility

The following macros are supported only for backward compatibility. There is no warranty that they will be supported in the next version of Processor Expert.

- **$REDDIR** - full path name of project directory for redirection
- **$REDINP** - full path name of input file for redirection
- **$REDOUT** - full path name of output file for redirection
- **$REDERR** - full path name of error file for redirection
- **$IN?DIR** - directory of input file, "?" is a number of input file from interval 0..9
- **$IN?NAM** - name of input file, "?" is a number of input file from interval 0..9
- **$IN?EXT** - extension of input file, "?" is a number of input file from interval 0..9
- **$IN?PTH** - full path name of input file, "?" is a number of input file from interval 0..9
- **$OUTDIR** - directory of output file
- **$OUTNAM** - name of output file
- **$OUTEXT** - extension of output file
- **$OUTPTH** - full path name of output file
- **$DRIVERS(FORMAT)** - list of the all generated drivers
- **$EVENTS(FORMAT)** - list of all generated event modules
- **$SHARED(FORMAT)** - list of all generated shared modules

2.2. Help and Manuals

Help | Processor Expert >

The following items are available within this menu:

- **Processor Expert Help** - the start page of the Processor Expert plug-in help.
- **Concepts** - introduction to Processor Expert concepts.
- **Benefits** - who Processor Expert may benefit.
- **User Interface** - description of Processor Expert plug-in environment.
- **Tutorial** - tutorial course.
- **Quick Start** - how to start with Processor Expert plug-in.
- **Embedded Beans** - index page of the Embedded Beans documentation.
- **Bean Categories** - index page of the Embedded Beans Categories.
- **Bean Keywords** - page of the commonly used keywords related to embedded beans.
- **Supported CPUs, Compilers and Debuggers** - list of CPUs/Compilers/ Debuggers supported in the current version of Processor Expert plug-in.
- **View Readme and Revision History** - Displays information about used Processor Expert plugin version, basic installation instructions, content of installation, FAQ, history of the signifiacnt changes from previous versions, known problems and limitations and other related information.
- **User Guide** - Opens a brief user's guide delivered with Processor Expert.

- **Search in PDF Documentation of the Target CPU** - displays PDF documentation of the current CPU in the PDF Search window. It is possible to search any keyword in the CPU documentation based on the original manufacturer's CPU manual. See chapter *2.17 PDF Search* for details.
- **Go to Processor Expert Home page** - display Processor Expert home page in default Internet browser.
- **Processor Expert On-line Support** - opens the web pages related to the customer support for the currently run Processor Expert version.
- **About Processor Expert & Tip Of The Day** - displays the **About dialog** containing information about the Processor Expert product version for the target CPU family and current version of Processor Expert IDE. The **Tip of the day** is displayed along with this dialog. Next tips can be viewed with using the button 'Next tip'.

  The **Installed updates** button opens the dialog containing all already installed update packages. After each update Processor Expert automatically copies installed update package to the folder shown by this dialog. Selecting an update package will show a window with detailed update description. The dialog is for information only and no action is done with the selected file.

2.3. Project Panel

Processor Expert | View | Project Panel

Processor Expert Project Panel is a tab in CodeWarrior's project window (panel). When the 'Project Panel' is noticed in Processor Expert documentation the 'Processor Expert Project Panel' is understood.

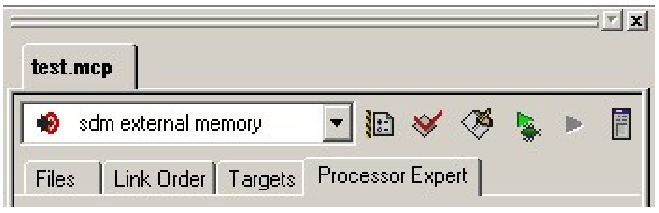


Figure 2.4 - Processor Expert tab

The **Project panel** shows the application components:

- **Configurations** of the project.
- Operating System - Beans related to operating system used by the application (if there is some operating system present).
- CPUs (**CPU beans**) included in the project
- **Embedded Beans** included in the project. Every bean inserted in the project is displayed in the project panel and has a subtree showing
  - **Methods** - Methods allow runtime control of the bean's functionality.
  - **Event routines** - Events allow handling of the hardware or software events related to the bean. If the event is disabled, the name of the event is shown. For enabled events, the name of the handling function is shown.
  - **Initialization code items** - Part of the initialization code that can/must be modified by the user.
  - **ISRs** - represent bean-related interrupt routines that can be created by the user for low level interrupt processing. For items, whose ISR names have been specified within a bean settings, a user-specified

name of an ISR and name of the interrupt vector is shown. If an ISR name was not specified (interrupt has to be disabled in this case), only the interrupt vector name is present.

- **PESL commands** - low-level PESL commands related to the peripheral configured by this bean. This folder is available only for Peripheral Initialization beans.

Notice: PESL is available only in the 56F800/E version.

All bean's items has its status icon that signalizes the enabled (☑) or disabled (☒) state. If this state cannot be directly changed, the background of the icon is gray. For more details please see chapter 3.2.1 Embedded Beans.

- **User modules** included in the project (main module, event module, external user modules ...)
- **Generated Modules** - This folder contains the modules generated by Processor Expert. There is a special subfolder for the generated Bean Modules. For the bean module description please see chapter 3.5.1 Code Generation.
- **External Modules** - This folder contains the modules that are not generated by Processor Expert but are required for the application such as libraries or system modules. These files are not influenced by Processor Expert but they are linked to the final application.
- **Documentations** - list of files attached into project as documentation, with relative or absolute path. No actions are made with these files. Please refer to chapter 3.5.1 Code Generation for details on generated documentation files.
- **PESL** methods (if PESL is enabled)

All Project Panel items are organized in folders in a tree. You can expand and collapse a tree's branches by clicking on the plus "+" or minus "-" signs, respectively. You can create your own folders in the ***Beans*** folder and **move beans between them** using mouse drag and drop function.

The following icons indicate the **status** of each project panel item:

✓	**Beans** - Processor Expert didn't found any problems in the bean's settings. **Configurations** - configuration is selected as active. **CPUs** - CPU is currently selected as a target CPU. **User Module, Generated Module** - The module is all right and included in the project. **PESL** - Processor Expert System Library is enabled.
✗	**Beans** - Gray cross means that bean is disabled and code won't be generated for it. **Configurations** - Configuration is not active. Double click the ✗ icon to select it as active configuration. **CPUs** - CPU is not selected as target CPU. Double click the ✗ icon to select it as the Target CPU. **User Module** - The user module is disabled. It is not possible to disable the Main and Event modules. **PESL** - Processor Expert System Library is disabled.
!	**Beans, CPUs** - beans (CPU bean's) settings are wrong or conflict with another bean. See chapter *3.3.3 Design Time Checking: Consequences and Benefits* for details.
?	Possibly incorrect setup was found in project or in bean's settings. The warnings are displayed in Error window including simple description.

Icons near the bean's icon mean the bean is individually setup for preserving user changes in generated code. See chapter *3.5.4 User Changes in Generated Code* for details.

The Project panel window allows **quick access** to supported **methods and events** using mouse. See paragraph Other mouse actions.

Pop-up menu

Project panel pop-up menu is accessible by right click on the empty (white) area of the Project Panel. Contains basic operations related to the project and beans.

- **Open Project** - allows the user to open Processor Expert project from disk.
- **Save Project** - saves the current state of the project (e.g. all the bean and processor expert settings). Project is also automatically saved when CodeWarrior or project is closed.
- **Copy Project to...** - copies the Processor Expert project file (.PE containing settings of all beans) and the user modules into another directory. It is useful for backing up the state of the project. The stored file could be opened again using **Open project** command from this menu.
- **Reload Project** - Reloads project from the last saved state on the disk.
- **Add Bean(s)** - allows to add beans from the project. Shows bean selector dialog.
- **Import...** - imports the content of the file containing exported objects (e.g. beans, configurations...) or whole project.
  All items from the imported file will be added into the current project.
- **Export...** - exports the selected objects in the project panel (e.g. beans, configurations...)to the specified file. Its possible to insert them into another project using the **Import...** command.
- **Cut** - cuts selected bean with its settings to the clipboard.
- **Copy** - copies selected bean with its settings to the clipboard.

- **Paste** - inserts bean from the clipboard to the current project.
- **Help** - displays related information for currently selected bean or method. If there is nothing selected this help page for Project Panel is displayed.

Pop-up Menus of Objects

Pop-up menus of individual objects in project panel are accessible with **a right mouse button click on the object's icon or label**.

- **Configurations pop-up menu**
- **CPUs pop-up menu**
- **Beans and its methods/events/init code pop-up menus**
- **User, generated and external modules menus**
- **Documentations pop-up menu**
- **PESL pop-up menu**

Other mouse actions

Drag'n'drop

- **Dragging method (or PESL command) with the left mouse button** to the source editor will place a method call to the source code. If the shift key is hold while the users drag and drops the method, the call is placed exactly to the mouse cursor position on the line. Otherwise the call is placed on the new line. A behavior of this function is controlled by the option **Environment Options | Drag'n'drop method declaration**. See chapter *2.1.1 Processor Expert Options* for details.
- **Dragging user module** into the source code will create an #include command (#include "user_module_name.h") at the place of the cursor.
- The user can drag'n'drop **components within the Project Panel** to reorganize component trees (CPUs, Beans, Documentation)

Multiselect

- Using the **Ctrl and Shift key** together with cursor key or left mouse button allows to select multiple items.

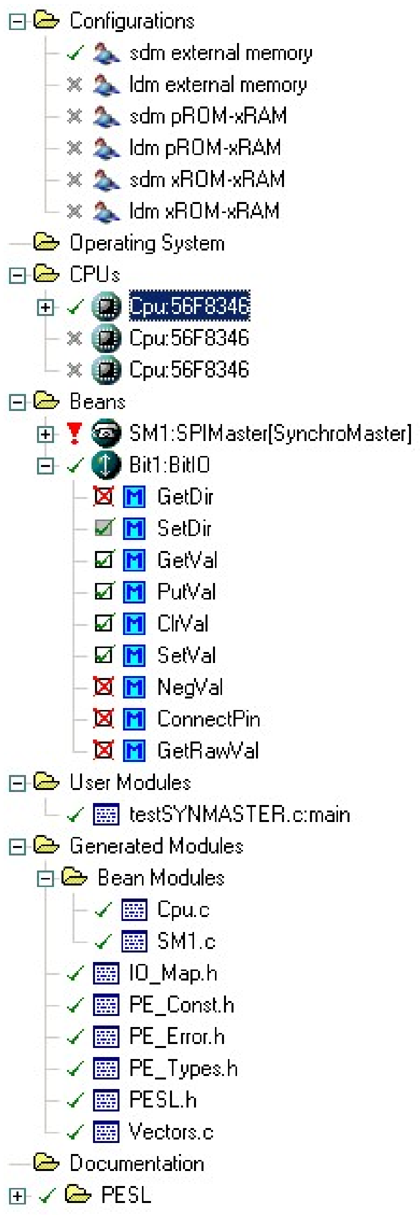
Double click

- Double-clicking the **bean icon** in the Project panel opens the **Bean Inspector**
- **Clicking on the selected bean name** in the Project Panel allows you to edit the name of the selected bean
- Double clicking on any event/method/initialization **enable/disable icon** changes its **enable/disable state** (you can do it also via the bean inspector)
- Double clicking on any **event/method name after code generation** opens the **file editor/viewer** at the position of the event/method's code
- Double clicking on any **ISR** opens the source code editor at the interrupt routine (if its name has been specified within the bean properties).

Automatic Hints

- Placing the cursor on any **event/method icon/name** displays the event/method's and parameter's description and syntax

- Placing the mouse over any **bean icon/name** displays the bean description



```
Configurations
  ✓ sdm external memory
  ✗ ldm external memory
  ✗ sdm pROM-xRAM
  ✗ ldm pROM-xRAM
  ✗ sdm xROM-xRAM
  ✗ ldm xROM-xRAM
Operating System
CPUs
  ✓ Cpu:56F8346
  ✗ Cpu:56F8346
  ✗ Cpu:56F8346
Beans
  ! SM1:SPIMaster[SynchroMaster]
  ✓ Bit1:BitIO
      ☒ GetDir
      ☑ SetDir
      ☑ GetVal
      ☑ PutVal
      ☑ ClrVal
      ☑ SetVal
      ☒ NegVal
      ☒ ConnectPin
      ☒ GetRawVal
User Modules
  ✓ testSYNMASTER.c:main
Generated Modules
  Bean Modules
    ✓ Cpu.c
    ✓ SM1.c
  ✓ IO_Map.h
  ✓ PE_Const.h
  ✓ PE_Error.h
  ✓ PE_Types.h
  ✓ PESL.h
  ✓ Vectors.c
Documentation
✓ PESL
```

Figure 2.5 - The content of the Project Panel

2.3.1. Configurations Pop-up Menus

Configurations folder pop-up menu

This menu is opened by right-clicking on the *Configurations* folder icon in the Project panel . Following commands are available:

- **Add new configuration** - add a new configuration into the project. All configuration settings (i.e. target CPU selection and state of all beans) are copied from the currently active configuration to the new one.
- **Configurations Editor** - opens the Configuration Editor.
- **Expand/Collapse** - expands/collapses one level of the folder's tree.
- **Expand all** - completely expands the folder's tree.
- **Collapse all** - completely collapses the folder's tree.
- **Delete all configurations** - deletes all the content of the folder.
- **Help** - displays documentation.

Configuration pop-up menu

This menu is opened by clicking on the icon of one of the configurations in the configurations folder of the Project panel. Following commands are available:

- **Configuration Inspector** - invokes Configuration Inspector (default on double-click) .
- **Select Configuration as Active** - selects this configuration as active.
- **Delete Configuration** - removes this configuration from the project.
- **Add New Configuration** - adds a new configuration to the project.
- **Rename Configuration** - renames this configuration.
- **Help** - displays documentation.

For more information about configurations see chapter 3.3.2 Configurations.

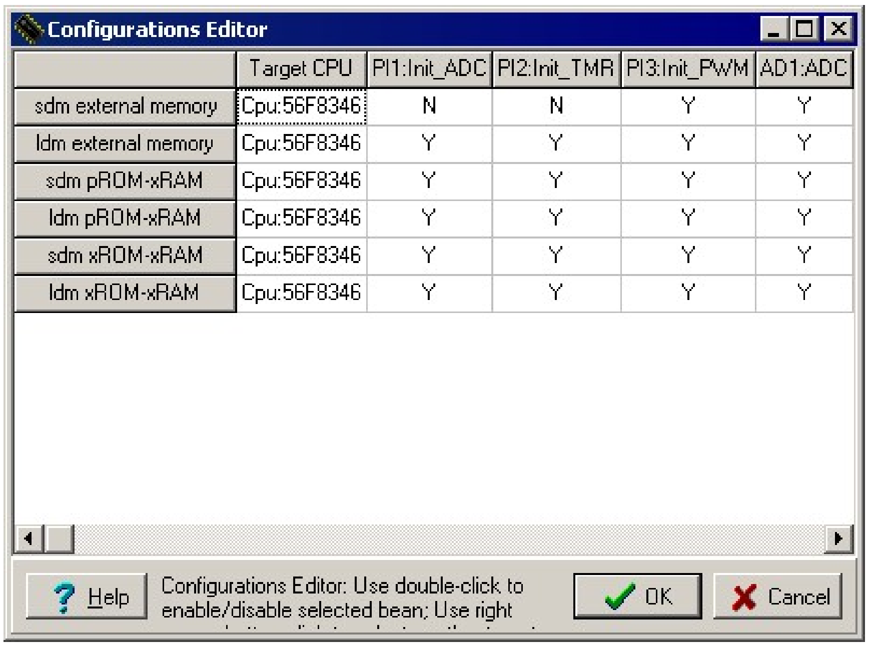
2.3.2. Configurations Editor

Project panel | Configurations pop-up menu | Configurations Editor

Configurations editor shows a table of all configurations and currently selected target CPUs and Embedded Beans state in the configurations. The leftmost gray column contains a name of one configuration in each row and the header of the table contains the names of the beans. For details on configurations please see the chapter 3.3.2 Configurations.

The first column of the table data contains the name of the currently selected target CPU for each configuration. To choose a different target CPU use the pop-up menu available after the right mouse button click on the appropriate cell.

Next columns represent the beans in the project. Each field in the beans columns contains a 'Y' if the bean is enabled or 'N' if the bean is disabled in the specific configuration. To change the value of the field from 'Y' to 'N' or vice versa double click the field with the left button.



	Target CPU	PI1:Init_ADC	PI2:Init_TMR	PI3:Init_PWM	AD1:ADC
sdm external memory	Cpu:56F8346	N	N	Y	Y
ldm external memory	Cpu:56F8346	Y	Y	Y	Y
sdm pROM-xRAM	Cpu:56F8346	Y	Y	Y	Y
ldm pROM-xRAM	Cpu:56F8346	Y	Y	Y	Y
sdm xROM-xRAM	Cpu:56F8346	Y	Y	Y	Y
ldm xROM-xRAM	Cpu:56F8346	Y	Y	Y	Y

Configurations Editor: Use double-click to enable/disable selected bean; Use right

2.3.3. CPUs Pop-up Menus

CPUs Folder Pop-up Menu

This menu is opened by right-clicking on the *CPUs* folder icon in the Project panel. It proposes a set of commands to manage CPUs folder. Following commands are available:

- **Expand/Collapse** - expands/collapses one level of the folder's tree.
- **Expand all** - completely expands the folder's tree.
- **Collapse all** - completely collapses the folder's tree.
- **Change folder name** - edit the subfolder name.
- **Delete folder** - deletes the subfolder and all its contents.
- **Delete all beans** - deletes all the contents of the folder.
- **Add subfolder** - creates a new subfolder.
- **Help** - Displays related information for currently selected bean or method. If there is nothing selected this help page for Project Panel is displayed.

CPU Pop-up Menu

This menu is opened by right-clicking on a CPU icon in the CPUs folder in the Project Panel.
Following commands are available:

- **CPU inspector** - opens the CPU's Bean inspector. Detailed help concerning the Bean Inspector items can be found in the CPU's page of Processor Expert help (see **Processor Expert | Help | Supported CPUs and Compilers** page).
- **Select CPU as Target** - if several CPUs are in the current project, it sets the CPU as target - the CPU will appear on the Target CPU window and the code will be generated with the Project options of active target.

- **Rename CPU** - allows you to give a project-specific name to the selected CPU
- **CPU Peripherals Names** - opens the CPU peripheral´s names editor.
- **View Target CPU Package** - opens the Target CPU window in Package view.
- **View Target CPU Block Diagram** - opens the Target CPU window in Block Diagram view.
- **View CPU Timing Model** - displays the CPU Timing Model window.
- **View Memory Map** - displays the Memory map window. This window shows the CPU address.

Space and internal and external memory mapping.

- **Search in PDF Documentation** - displays the PDF Search window. The window allows a full-text search in the original CPU manufacturer's documentation. See chapter *2.17 PDF Search* for details.
- **View Source** - displays the generated CPU module in the File editor.
- **View/Edit Event Module** - displays the generated CPU events module in the File editor.
- **View/Edit Main Module** - displays the generated main module in the File editor.
- **View Linker File** - displays the generated linker file (if it exists) in the File editor.
- **View Makefile** - displays the generated maker file (if it exists) in the File editor.
- **View MAP File** - displays the MPA file in the File editor.
- **View List of Methods** - displays the list of methods.
- **Restore Default Template Settings** - restores default setting of the template.
- **Customize this bean template** - saves the highlighted (chosen) CPU and its settings as a template.
- **Remove CPU From Project** - removes the highlighted (chosen) CPU from the project.
- **Help** - displays documentation.

2.3.4. Beans Pop-up Menus

Beans Folder Pop-up Menu

This menu is opened by right-clicking on the *Beans* folder icon in Project Panel. Following commands are available:

- **Expand/Collapse** - expands/collapses one level of the folder's tree.
- **Expand all** - completely expands the folder's tree.
- **Collapse all** - completely collapses the folder's tree.
- **Change folder name** - edits the subfolder name.
- **Delete folder** - deletes the subfolder and all its contents.
- **Delete all beans** - deletes all beans from the project.
- **Add bean(s)** - invokes a dialog which allows the user to choose and add new beans to project.
- **Add subfolder** - creates a new subfolder.
- **Import** - Imports all items from a .pe file containing exported objects or whole project.
- **Export** - Exports selected objects in the project panel to the file.
- **Help** - displays documentation.

Bean Pop-up Menu

This menu is opened by clicking right mouse button on the icon of the bean from the beans folder of the Project panel. Following commands are available:

- **Bean inspector** - opens the Bean inspector of the bean. Detailed help concerning the Bean Inspector items can be found in the bean's page of Processor Expert help.
- **Bean enabled** - if checked, the selected bean is enabled (included in project).
- **Code generation** - allows to individually specify how the and user changes and code generation for the bean are handled by Processor Expert. See chapter *3.5.4 User Changes in Generated Code* for details.
  - **Always Write Generated Bean Modules** (default) - generated bean modules are always written to disk and any existing previous module is overwritten
  - **Preserve User Changed in Generated Bean Modules** - smart detection of user changes.

    Notice: Smart user changes preservation is available only in the 56F800/E version.

  - **Don't Write Generated Bean Modules** - the code from bean is not generated. Any initialization code of the bean, which resides in the CPU bean, interrupt vector table and shared modules are updated.
  - **Compare with Previously Generated Module, Compare with Previously Generated Header Module** - compares a file generated by the bean with a previously generated one. The user can use this function to easily track his/her changes made in the generated code. The both commands are available only when the 'Preserve User Changes' option is switched. See chapter *2.1.1 Processor Expert Options* for details. The bean has to be setup to 'Preserve User Changed in Generated Bean Modules' or 'Don't Write Generated Bean Modules' mode (in this pop-up menu). An internal file-editor in read-only comparison mode is used to show the files differences. See chapter *2.16 File Editor* for details.
- **Rename Bean** - allows you to give a project-specific name to the selected bean.
- **View Source** - displays the generated module of the bean in File editor.
- **View/Edit Event Module** - displays the generated events module of the bean in the File editor.
- **Restore Default Template Settings** - restores default template settings. All old settings will be lost.
- **Customize this bean template** - saves the selected bean as a template
- **Disconnect Bean From CPU** - removes the link(s) between the bean and the associated CPU peripheral(s) (it clears the corresponding properties of the bean).
- **Remove Bean From Project** - removes the selected bean from the current project.
- **Copy to Clipboard** - bean with its settings is copied to the clipboard.
- **Help** - displays documentation.

Methods and Events

This menu is opened by right-clicking on a method or event icon in the Project Panel. It proposes a set of commands concerning the selected method/event.

- **Enable/Disable** - enables/disables the selected method/event in current project.

- **View source***(method only)* - shows generated method source. Available only after successful code design.
- **Edit code***(event only)* - opens a selected event in editor. Available only after successful code generation.
- **Help** - displays documentation.

Init Code

This menu is opened by right-clicking on init code item on its icon in the Project Panel. It proposes a set of commands concerning the selected initialization code item.

- **Enable/Disable** - enables/disables the selected item in current project.

- **View source** - shows generated source of the initialization. Available only after successful code generation. Please see the code and comment describing what should be provided in user's code.
- **Help** - displays documentation.

ISRs

This menu is opened by righ-clicking on ISR item in the Project Panel.

- Rename ISR - opens the Bean Inspector and selects the property that specifies the name of the interrupt routine.
- Edit Code - opens the source code editor at the interrupt routine (if its name has been specified within the bean properties).
- Help - Shows the related help for the bean.

2.3.5. User and Generated Modules Pop-up Menus

User Modules Folder Pop-up Menu

- **Add User Module** - This menu allows to add a user source code module of the specified type to the project. The user can choose the file extension from the submenu and select the file using a standard windows file-selection dialog.
- **New User Module** - User can choose a new file type and specify the file name and path. The new file is added to the project.
- **Expand/Collapse** - expands/collapses one level of the folder's tree
- **Expand All** - completely expands the folder's tree.
- **Collapse All** - completely collapses the folder's tree.
- **Delete All User Modules** - removes all previously added user modules from the project. The user is asked for a permission on removing the modules. The main user module *{projectname}.c* and events module

events.c cannot be removed from the project.

- **Help** - displays an appropriate help page.

User Module Pop-up Menu

- **Edit Source...** - opens the source code in the editor.
- **User Module Enabled** - The user module is enabled for the compilation.
- **User Module Inspector** - The *User Module Inspector* window is shown. It allows the user to customize the name and directory of the module.
- **Remove User Module** - removes the user module from the project.
- **Help** - displays an appropriate help page.

Generated Modules Pop-up Menu

- **View Source** - opens the module source code in the editor.
- **Code Generation** - enables/disables overwriting of the module by Processor Expert. This option is available only for common modules (like vectors.c) or modules not related to a specific bean. For bean modules generation control use the pop-up menu of the bean instead. See chapter *3.5.4 User Changes in Generated Code* for details.
  - **Always Write** - allows a module to be overwritten.
  - **Don't Write** - disables any modification by Processor Expert.
- **Help** - displays an appropriate help page.

External Modules Pop-up Menu

- **View External Module** - opens the module source code in the editor.
- **Help** - displays an appropriate help page.

2.3.6. Documentations Pop-up Menu

Documentations Folder Pop-up Menu

This menu is opened by right-clicking on the *Documentation* folder icon in the Project Panel. Following commands are available:

- **Add documentation file** - add a new documentation file into project
- **New documentation**
  - **Text file** - creates a new text documentation file.
  - **HTML file** - creates a new HTML documentation file.
- **Expand/Collapse** - expands/collapses one level of the folder's tree.
- **Expand all** - completely expands the folder's tree.
- **Collapse all** - completely collapses the folder's tree.
- **Delete all documentation** - deletes all the contents of the folder.
- **Help** - displays documentation.

Documentation Pop-up Menu

This menu is opened by clicking right mouse button on the documentation file from the Documentations folder of the Project panel. Following commands are available:

- **View documentation** - shows the document in the File Editor.
- **Edit documentation** - edits the document in the File Editor.
- **Open in external viewer** - uses the default shell editor to open the document.
- **Remove from project** - removes the document from the project.
- **Help** - displays documentation.

2.3.7. PESL Folder Pop-up Menus

PESL Folder Pop-up Menu

This menu is opened by right-clicking on the *PESL* folder icon in the Project Panel. Following commands are available:

- **Expand/Collapse** - expands/collapses one level of the folder's tree.
- **Expand all** - completely expands the folder's tree.
- **Collapse all** - completely collapses the folder's tree.
- **PESL enabled** - enables/disables use of the PESL (Processor Expert System Library)
- **Help** - displays documentation.

PESL Command Pop-up Menu

This menu is opened by right-clicking on the PESL item.
Following commands are available:

- **Help** - displays PESL documentation for the selected item.

2.4. Bean Selector

Processor Expert | View | Bean Selector

Bean Selector shows supported embedded beans including CPU beans and bean templates. It lets the user select a desired bean or template and add it to the project.

Two filters could be applied on the bean list. They could be switched on/off by clicking on two buttons on the bottom bar.

- **All/CPU** - If this filter is active, only the beans that could be used with the current target CPU derivative are shown.
- **Licensed** - If active, only the beans with valid license are shown.

The Bean Selector contains the following four tabs allowing the user to see list of the bean in the following modes:

- **Bean Categories** - contains all available beans. The beans are sorted in a tree based on the categories defined in the beans. See chapter *3.2.1.1 Bean Categories* for details.
- **On Chip Peripherals** - shows all beans available for the specific peripherals. All chip peripherals, sorted by name, are listed in the appropriate CPU folder, depending on which peripheral can be used. Current target CPU bean is displayed at the top (only if a target CPU bean is selected).
  There are three different icons of peripheral folders which depends on the usage of the peripheral.

  - If the peripheral is **fully available**, the folder is displayed by yellow icon.
  - If the peripheral is **partially used**, the folder is displayed by light blue icon.
  - The **fully used** peripheral is displayed by blue icon.

  the Bean Selector provides the "**On Chip Peripherals**" view for the users, that are not familiar with the beans functionality yet (but they know the chip peripherals). This page contains all on-chip peripherals of the selected CPU and for each peripheral list of supported beans. So it's very easy to find bean, that supports functionality of the selected peripheral.

- **Alphabet** - shows alphabetical list of available beans. The user can speed-up searching the right bean **typing the start of the bean name on the keyboard**. All/CPU and license filters could be used here like on the other Bean Selector tabs.

- **Keywords** - shows alphabetical listing of keywords related to the internal peripherals. The list of available beans that could use the keyword-related peripheral can be found under each keyword. All/CPU and License filters are used here as well.

The icon means that there is an available license for the bean. If the icon is displayed as a "greyed" icon, it means that for the selected bean a valid license is not available.

The bean names are colored black and the bean template names are colored blue. By **double-clicking on the bean** it is possible to insert the bean into the current project. The description of the bean is shown in a hint.

The button **Quick Help** shows short information about function of the bean. The Quick Help is displayed as a part of the Bean Selector window and is updated when the user selects another bean in the tree.

Folder Pop-up menu

The pop-up menu is available by clicking the right mouse button on a folder.

- **Expand/Collapse** - expands or collapses the folder
- **Expand all** - expands the folder and all its subfolders
- **Collapse all** - collapses the folder and all its subfolders
- **Help on Bean Selector**- displays documentation for the Bean Selector.

Bean Pop-up Menu

The pop-up menu is available by clicking the right mouse button on a bean.

- **Add the bean to the current project** - adds the bean to the current project.
- **Delete selected template**- removes the selected template from the Bean Selector.
- **Help on Bean**- displays bean documentation.
- **Help on Bean Selector**- displays documentation for the Bean Selector.

Bean Selector Pop-up Menu

The pop-up menu is available by clicking right mouse button on the area inside the Bean Selector window

- **Update** - updates new beans and templates to the tree according to the appropriate category in the Bean Selector window.
- **Help on Bean Selector** - displays documentation for the Bean Selector.

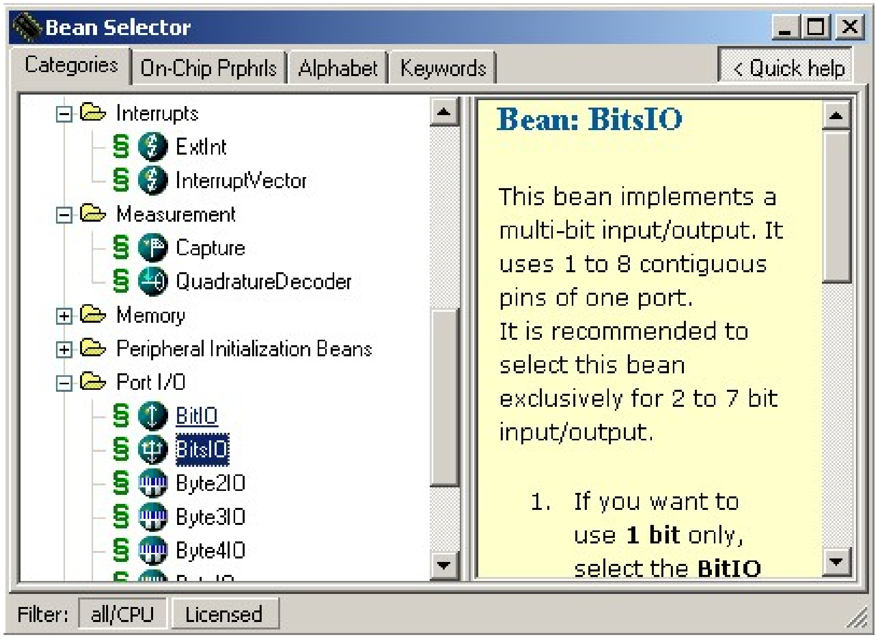
Target CPU Folder Pop-up Menu

The pop-up menu is available by clicking the right mouse button on the Target CPU folder in the On Chip Peripheral mode. This menu is the same as the pop-up menu for the target CPU in the project panel. See chapter *2.3 Project Panel* for details. for details.

Peripheral Folder Pop-up Menu

The pop-up menu is available by clicking right mouse button on the peripheral in the On Chip Peripheral mode.

- **Expand/Collapse** - expands or collapses the folder
- **Expand All** - expands the folder and all of its subfolders
- **Collapse All** - collapses the folder and all of its subfolders
- **Show Peripheral Structure** - opens the peripheral's structure view - (it is supported for I/O ports, timer's counters, serial ports. It is also supported for devices working in several modes in the CPU block diagram. A list of represented devices for these modes is displayed.
- **Rename Peripheral** - allows the user to rename the selected peripheral. It is supported for I/O ports and pins, watchdog and timers (counters, compare and capture registers, free running devices), A/D converters and A/D channels, CAN, serial ports. See details for renaming peripherals.
- **Show Peripheral Usage** - shows which part of the peripheral is used by the application (visible after code generation). It is supported for I/O ports and pins, timers, A/D converters and A/D channels, CAN, serial ports, watchdog, internal memories (EEPROM and FLASH). See chapter *2.15 Peripherals Usage* for details.
- **Show Peripheral Initialization** - shows initialization values of all "control, status and data" registers. It is supported for all devices displayed on CPU package. See chapter *2.14 Peripheral Initialization* for details.
- **Search Related Info in CPU PDF Documentation** displays the PDF Search window and finds information about the peripheral in the appropriate CPU documentation. It is for possible to search for any keyword in the CPU documentation based on the original manufacturer's CPU manual. (This item is available on the package and on the CPU block only.) See chapter *2.17 PDF Search* for details.
- **View CPU Block Diagram** - displays the CPU block diagram in the Target CPU window.
- **Help on Bean Selector** - displays documentation on the Bean Selector



Bean: BitsIO

This bean implements a multi-bit input/output. It uses 1 to 8 contiguous pins of one port.
It is recommended to select this bean exclusively for 2 to 7 bit input/output.

1. If you want to use **1 bit** only, select the **BitIO**

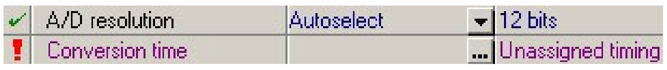
Figure 2.7 - Bean Selector with Quck Help panel

2.5. Inspector

Processor Expert | View | Inspector

Inspector is universal window, which allows to view and edit attributes of the object selected in the Project Panel. It could be a Bean, Configuration, User module or Peripheral Initialization bean. Inspector can work in these modes depending on the type of inspected object.

- **Bean Inspector** - provides access to Properties, Methods, Events, and Comments for the beans, arranged in switchable pages. See chapter *2.5.3 Bean Inspector* for details. Bean Inspector for CPU bean offers additional Build options (if a target compiler is selected) and Used peripherals pages.
- **Configuration Inspector** - Provides access to settings of a configuration. See details in chapter Configuration Inspector.
- **User module inspector** - Provides access to settings of a user module.



✓	A/D resolution	Autoselect	12 bits
!	Conversion time	...	Unassigned timing

Figure 2.8 - Example of the Inspector Window content

Window Columns

Inspector window contains the four columns:

- **Item status**
  - ✓ green checkmark - item setting is correct
  - ❗ red exclamation - item setting is not correct. Items that cause errors or warnings are written in magenta color. See description in the last column or the Error Window.
  - ⊞ plus or ⊟ minus - item is a group of settings that can be expanded/collapsed.
  - ✓ light background - item is version specific. See chapter *2.5.3.3 Version Specific Items* for details.
- **Item names** - items that are to be set are listed in the second column of the inspector. Groups of items describing certain features may be collapsed/expanded by double clicking on the first line of the group. By double clicking on a method or event item, you may open the File Editor at the position of the corresponding method or event.
- **Selected settings** - the settings of the items are made in the third column. See chapter 2.5.1 Inspector Items for list of item types.
- **Setting status** - the current setting or an error status may be reflected on the same line, in the rightmost column of the inspector.

Read only items

Any item can be presented as read-only so the user could not change its content. Read only values are gray.

✓	Width	9 bits	9 bits

Menu

The following items are available:

- **Bean** *(enabled in Bean Inspector Only)*
  - **Template**
    - **Restore default template settings** - restores settings of the template.
    - **Save bean settings as template** - invokes template editor. See details on Bean Templates here.
    - **Active template** - Shows list of currently available templates for the bean with currently active template selected.
  - **Change bean icon** - allows the user change the bean icon.
  - **Autoconnect** - auto connects the bean to the CPU.
  - **Disconnect** - disconnects the bean from the CPU.
- **Items Visibility** - see the chapter Items Visibility for more information about view modes.
- **Help**
  - **Help on Selected tab** - displays documentation for the current tab.
  - **Help on Inspector** - displays Bean Inspector documentation.
  - **Help on Bean***(enabled in Bean Inspector Only)* - displays documentation for the selected bean.
  - **Embedded Beans Page** - displays Embedded Beans documentation.

- Navigation buttons ‹ › allow the user to browse over the previously inspected beans.
- **Peripheral initialization button** *(present in Bean Inspector Only)* - This button will show the Peripheral initialization. If the button is pushed, the peripheral initialization window is attached to the Bean inspector window.

View mode buttons

They are placed at the bottom of the window (Basic, Advanced, Expert). They allow users to switch complexity of the view of the bean's items. See chapter *2.5.2 Items Visibility* for details.

Pop-up Menu

This menu is invoked by a click of the right mouse button on the specific inspector item. The menu contains the following commands:

- **New Item Into List** - adds a new item before the currently selected one. This item is available only for the list-type properties.
- **Delete Item From List** - deletes a selected item from the list. This item is available only for the list-type properties.
- **Move List Item Up** - Moves the selected row towards the start of the list. This item is available only for the list-type properties.
- **Move List Item Down** - Moves the selected row towards the end of the list. This item is available only for the list-type properties.
- **Help on the Item** - shows the appropriate help page for the selected item.

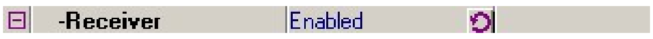
2.5.1. Inspector Items

The following types of the items could be found in the Inspector

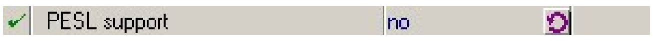
Alphabetical list

Descriptions

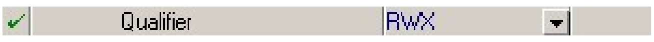
- **Boolean Group** - A group of settings controlled by this boolean property. If the group is enabled, all the items under the group are valid; if it is disabled, the list of items is not valid. Clicking the + sign will show/hide the items in the group but doesn't influence value nor validity of the items.


-Receiver Enabled

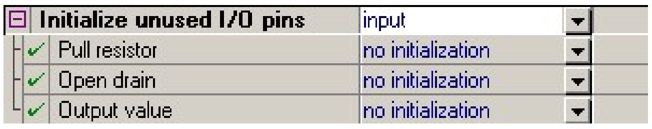
- **Boolean yes / no** - The user can switch between two states of the property using a round icon .
  The **Generate code / Don't generate code** settings of methods and events works the same way and determines whether the implementation code for the corresponding method or event will be generated or not (you may thus generate only the methods and events used by your application).


PESL support no

- **Enumeration** - Selection from a list of values. If the user clicks the arrow icon (▾), a list of the possible values for the property is offered.

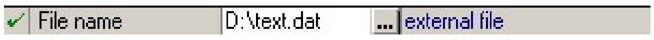

Qualifier RWX

- **Enumeration Group** - A list of items. Number of visible (and valid) items in the group depends on chosen value. Clicking the arrow icon ( ) will show a list of the possible values of the property. Clicking the + sign will show/hide the items in the group but doesn't influence value nor validity of the items.



Initialize unused I/O pins	input
Pull resistor	no initialization
Open drain	no initialization
Output value	no initialization

- **File/Directory Selection** - allows to specify a file or directory. Clicking the icon will open a system dialog window allowing to choose a file/directory.

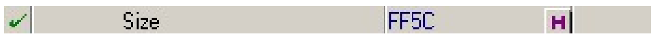


File name	D:\text.dat ...	external file

- **Group** - A list of items which can be expanded/collapsed by clicking on the plus/minus icon or by double clicking at the row. Values of the items in the group are untouched.

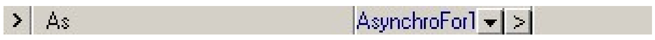
+Initialization

- **Integer Number** - The user can insert a number of a selected radix. Radix of the number could be switched using the icons (D = Decimal ,H = Hexadecimal, B = Binary). Only reasonable radixes are offered for the property. If the radix switching icon is not present, Processor Expert expects the decimal radix.



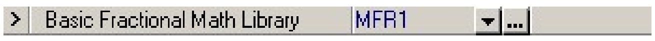
Size	FF5C H

- **Link to inherited bean** - The arrow icon switches the inspector to the ancestor bean that is inherited by the current bean. The down-arrow button allows to change the ancestor from the list of possible ancestor. See chapter *3.3.7 Bean Inheritance and Bean Sharing* for details.



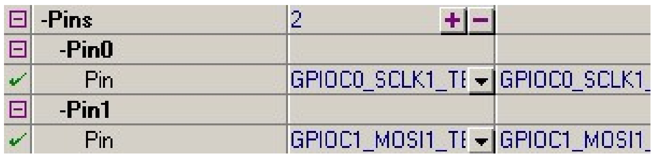
As	AsynchroFor1

- **Link to shared bean** - The dialog button switches the inspector to the shared bean that is used by the current bean. The down-arrow button allows to change the bean from the list of the available beans. See chapter *3.3.7 Bean Inheritance and Bean Sharing* for details.



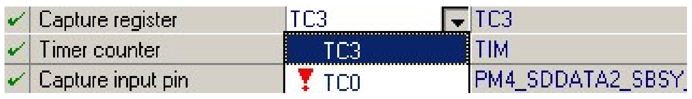
Basic Fractional Math Library	MFR1

- **List of items** - A list of items may be expanded/collapsed by clicking on the plus/minus button in the left side of the row or by double clicking on the row. The user may add/remove items by clicking on the plus/minus button. The items in the list can be arranged using a related pop-up menu commands.



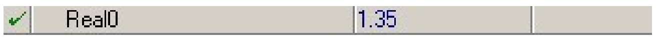
-Pins	2 + −	
-Pin0		
Pin	GPIOC0_SCLK1_TE	GPIOC0_SCLK1_
-Pin1		
Pin	GPIOC1_MOSI1_TE	GPIOC1_MOSI1_

- **Peripheral selection** - The user can select a peripheral from the list of the available peripherals. The peripheral that are already allocated have the bean icon in the list. The properties that conflicts with the bean settings have the red exclamation mark.



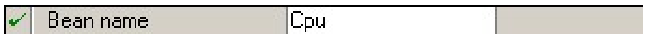
✓	Capture register	TC3	TC3
✓	Timer counter	TC3	TIM
✓	Capture input pin	TC0	PM4_SDDATA2_SBSY_

- **Real Number** - the user can insert any real (floating point) number.



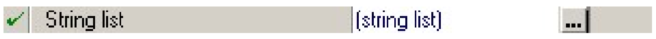
✓	Real0	1.35	

- **String** - Allows to enter any text or value



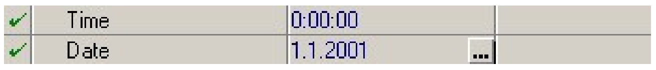
✓	Bean name	Cpu	

- **String list** - Clicking the dialog button ... will open the simple text editor that allows to enter an array of text lines.



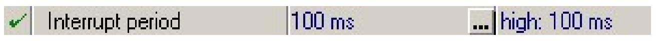
✓	String list	(string list)	

- **Time, Date** - Allows to setup the Time/Date in a format according to the operating system settings.



✓	Time	0:00:00	
✓	Date	1.1.2001	

- **Timing settings** - Allows a comfortable setting of the bean's timing. The timing dialog box gets opened when clicking on .... See chapter *2.5.3.1 Dialog Box for Timing Settings* for details.



✓	Interrupt period	100 ms	high: 100 ms

2.5.2. Items Visibility

Processor Expert supports **selectable visibility** of bean items. Each item is assigned a predefined level of visibility. **Higher visibility level** means that items with this level are more special and rarely used than the others with the lower visibility level. Bean Inspector displays only items on and below the selected level. It could help especially beginners to set only basic properties at first and do optimization and improvements using advanced and expert properties or events later. There are three visibility levels:

- **Basic view** - key items that have no default value and must be set for each bean, for example peripherals. To view these items select command **Inspector | Items Visibility | Basic view**.
- **Advanced view** - key items that should be set or that are usually set (including necessary items) To view these items select command **Inspector | Items Visibility | Advanced view**.
- **Expert view**- maximum visibility level. All items that can be changed including rarely used items. To view these items select command **Inspector | Items Visibility | Expert view**.

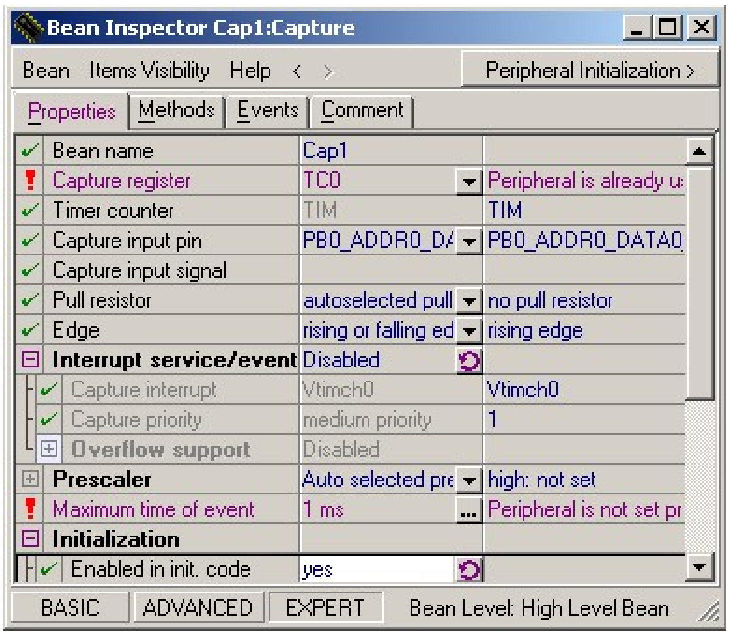
See also the main page of the Bean Inspector chapter for more information how to switch view modes.

Note: If an error occurred in a property with a higher visibility level than the level currently selected, this error nevertheless will be displayed.

2.5.3. Bean Inspector

Bean inspector is one of the Inspector window variants. It allows to setup **Properties**, **Methods**, and **Events** of a bean. Use command **Help | Help on Bean** from Bean Inspector menu to see documentation for currently opened bean.

Note: Property settings influencing the hardware can often be better presented by the CPU package view using Target CPU window. See chapter 2.7 Target CPU Window for details.


Bean Inspector Cap1:Capture
Bean
Items Visibility
Help
<
>
Peripheral Initialization >
Properties
Methods
Events
Comment
BASIC
ADVANCED
EXPERT
Bean Level: High Level Bean

✓	Bean name	Cap1	
!	Capture register	TC0	Peripheral is already u
✓	Timer counter	TIM	TIM
✓	Capture input pin	PB0_ADDR0_DA	PB0_ADDR0_DATA0
✓	Capture input signal		
✓	Pull resistor	autoselected pull	no pull resistor
✓	Edge	rising or falling ed	rising edge
⊟	Interrupt service/event	Disabled	
✓	Capture interrupt	Vtimch0	Vtimch0
✓	Capture priority	medium priority	1
⊞	Overflow support	Disabled	
⊞	Prescaler	Auto selected pre	high: not set
!	Maximum time of event	1 ms	Peripheral is not set pr
⊟	Initialization		
✓	Enabled in init. code	yes	

Figure 2.26 - Bean Inspector Window

The **Comments** page allows the user to write any comment about the bean or setting used. This comment will be displayed in the hint when the mouse cursor will be placed on the bean.

The **Build options** page is present **only in the CPU bean** and it provides access to the settings of the compiler (or debugger) selected in Project Options. These settings are different for each compiler (or debugger), and are reset every time the compiler (or debugger) is changed.

The **Used** page shows list of the CPU bean resources. The user can also manually block individual resources for using them in Processor Expert.
The page consists of the three columns:

- First shows the name of the resource. Resources are in groups according to which device they belong to.
- Second column allows the user to reserve resource (for example pin) for external module. Click on icon to reserve/free a resource. **Reserved resource could not be use in Processor expert any more.**
- Third column shows the current status of the resource and the name of the bean which uses it (if the resource is already used).

For **menu and view mode description** and other common Inspector window features see chapter 2.5 Inspector and 2.5.1 Inspector Items.

Pin sharing

Some beans allow sharing of the pins. This ability is indicated by a presence of the pin sharing button in the pin selection property line. See chapter *3.3.8 Pin Sharing* for details.

Bean level

The Bean Level is displayed at the bottom of the window besides the view mode buttons. It describes the amount of the peripheral abstraction and a cross platform portability.

- **High Level Beans** - highest level of peripheral abstraction. An application built from these beans can be easily ported to another microcontroller supported by the Processor Expert.
- **Low Level Beans** - The beans that are dependent on the peripheral structure to allow the user to benefit from a non-standard features of a peripheral.
- **Peripheral Initialization Beans** - These beans are on the lowest level of abstraction. An interface of such beans is based on the set of peripheral control registers. These beans cover all features of the peripherals and were designed for initialization of these peripherals (contain only one method "Init" and no events).

Please see chapter 3.2.1.1 Bean Categories for more information.

2.5.3.1. Dialog Box for Timing Settings

The **Timing dialog** box provides a user-friendly interface for the settings of bean timing features. When clicking on the button of a timing item in the Bean Inspector, the timing dialog box is displayed.

Before you start to edit bean timing you should set:

- **Target CPU** in the Project Panel
- **Used peripherals** in the bean's properties
- **Requested prescaler** in the bean's properties, if it is supported and needed for the application.
- **Supported speed modes** in the bean's properties

The settings are instantly validated according to the Processor Expert timing model. For details on the timing settings principles please see the chapter 3.3.4 Timing Settings.

Options

Upper left panel
Runtime setting
determines how the timing setting can be modified in runtime.
Runtime setting is not supported in the BASIC view mode.
Runtime setting type:

- **fixed value**: it will not be possible to change the timing setting of the bean in runtime.
- **from a list of values**: it will be possible to select a new value among predetermined values from the list (use the *Add* and *Delete* buttons to control number of values in the list). Each value in the list defines a *mode* and you can switch between them using method *Set???Mode* (name of the method depends on the bean).
- **from interval**: it will be possible to select a new value within a predetermined interval.
  This Runtime setting type requires runtime calculations to change time setting of the bean. The runtime setting type may not be supported on small microcontrollers.

The modification of timing settings at runtime is done using Bean methods. Some of them will be enabled only if you select corresponding Runtime Setting Type.

Upper right panel

Value

currently edited time: initialization value, value in any mode from the list or limit of the interval.

- **Requested value**: write here the value you wish to set and select the unit in the **Units** list box. If you select the **With value** check- box, the value is recomputed at every unit change.
  Tip: Double- click on the requested value to see all supported values close to the selected one.
- It is possible to specify desired precision of the timer settings by using one of the following settings (which one is used depends on the type of the timing) :
  - **Error allowed** - This field allows specifying the tolerated difference between real timing and the requested value. The **%** check-box allows the user to set the degree of precision as a percentage of the requested value.
  - **Min. resolution** - Minimal resolution of timer ticks. This setting is used for setting interval or capture bean timing. Allows the user to specify maximal acceptable length of one tick of the timer. In the case of interval settings the **%** check-box (if it is present) allows the user to set the degree of precision as a percentage of the low limit value. Otherwise the **%** value is related to the requested value.
- **Adjusted value**, **Prescaler** and **Error:** Shows the real value computed from chosen on-chip peripheral settings, selected Prescaler value and the difference between the value selected by the user and the real value.
- **Status line** displays status of the timing setting. If the requirements are impossible to meet, a red error message is displayed.

Bottom panel

Possible settings

values supported by the Target CPU for the selected peripheral.

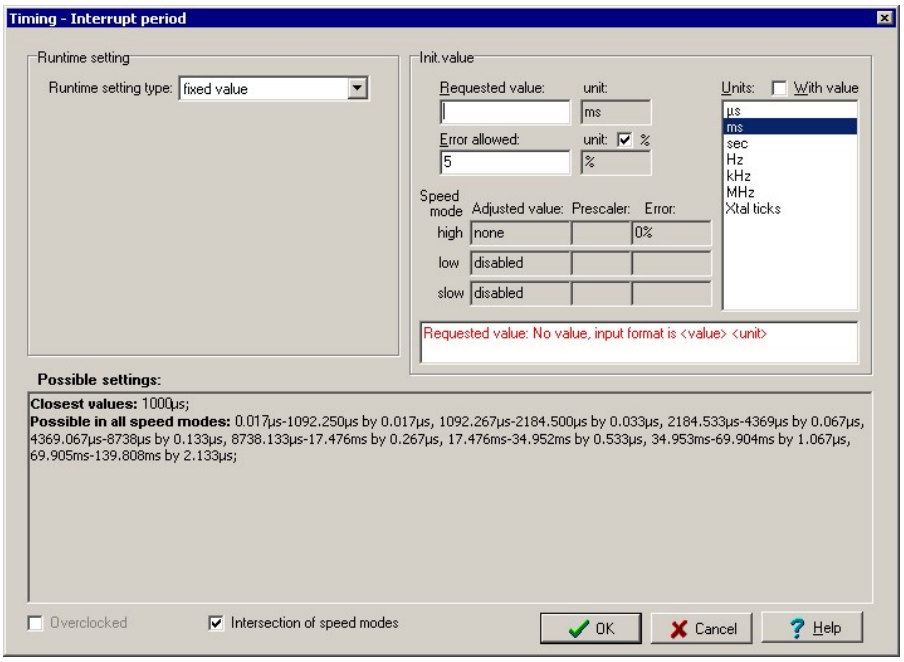
- **Closest values**: supported time values that are the closest to the requested value.
- **Possible in high (and/or low, slow) speed mode**: all correct settings in high/low/slow speed mode.
  Tip: Click on any value to see all supported values close to the selected one.
- **Overclocked** (check-box below for information only): if checked, the bean supports extension of the hardware timing by the 8-bit software counter.
- **Intersection of speed modes** (check-box below is visible only in EXPERT view mode) if selected the *Possible settings* box contains only values which are available in all speed modes supported by the bean.

Note: *Speed modes and related settings are supported only in EXPERT view mode.*

Prescaler

It is possible to set a requested prescaler value in several beans. You can choose "Autoselect" for automatic selection (default value).

Some peripherals support *internal prescalers* - any other peripheral can divide its clock. You can set this special prescalers in the bean properties - item Prescaler (if supported). The internal prescaler are not used by automatic selection. Internal prescaler peripherals can be selected in other beans as internal prescaler but cannot be used as a peripheral (as an interrupt source).



Runtime setting type: fixed value

Requested value:	unit:
	ms

Error allowed:	unit: ✓ %
5	%

Speed mode	Adjusted value:	Prescaler:	Error:
high	none		0%
low	disabled		
slow	disabled		

Requested value: No value, input format is <value> <unit>

Possible settings:

Closest values: 1000µs;
Possible in all speed modes: 0.017µs-1092.250µs by 0.017µs, 1092.267µs-2184.500µs by 0.033µs, 2184.533µs-4369µs by 0.067µs, 4369.067µs-8738µs by 0.133µs, 8738.133µs-17.476ms by 0.267µs, 17.476ms-34.952ms by 0.533µs, 34.953ms-69.904ms by 1.067µs, 69.905ms-139.808ms by 2.133µs;

✓ Intersection of speed modes

Figure 2.27 - Timing Settings Dialog

2.5.3.1.1. Syntax for the Setting of Timing Features in the Bean Inspector

In timing data fields, it is necessary to type not only a **value** (integer or real number) but also the **unit** of that value.

Supported units

- **microseconds** - the value must be followed by `us`
- **milliseconds** - the value must be followed by `ms`
- **seconds** - the value must be followed by `s`
- **CPU ticks** - the value must be followed by `CPU crystal/oscillator ticks`
- **Hertz** - the value must be followed by `Hz`
- **kilohertz** - the value must be followed by `kHz`
- **megahertz** - the value must be followed by `MHz`
- **bit/second** - the value must be followed by `bits`
- **kbit/second** - the value must be followed by `kbits`

Example

If you want to specify 100 milliseconds, enter `100 ms`

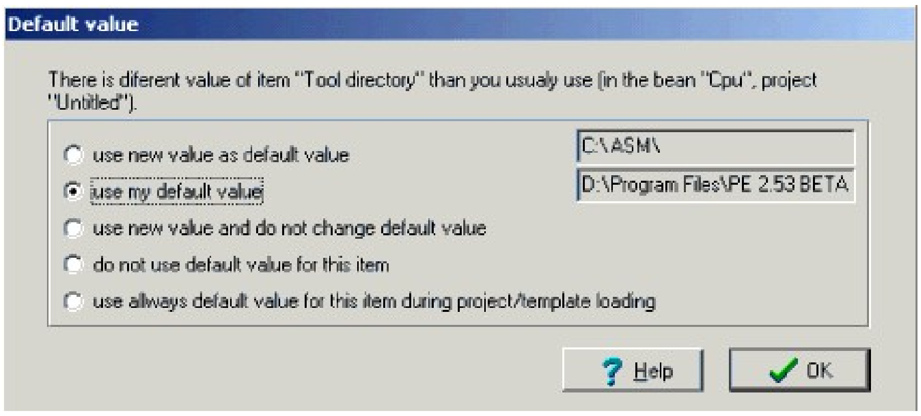
2.5.3.2. Defaut Values for Properties

Some properties can have a global default value. Once you enter the value for the property and you confirm that it will be the default value for the property then it will be used automatically in the future.

If you open a project with different settings or change the value of the property in the Bean Inspector, a Processor Expert automatically offers the following options:

- **use new value as default value** - remember the newly entered (or just loaded) value as the new default value (change default value),
- **use my default value** - cancel changes (discard loaded value) and use previous default value,
- **use new value and do not change default value** - use the newly entered (or just loaded) value and do not change the default value,
- **do not use default value for this item** - *permanent settings*, never use default value for this property,
- **use always default value for this item during project loading** - *permanent settings*, do not display this dialog and always use the default value during project loading and template creation.
  Note: This item is accessible only during project loading.

Permanent settings or default values can be removed in Environment Options.


Default value

There is diferent value of item "Tool directory" than you usualy use (in the bean "Cpu", project "Untitled").

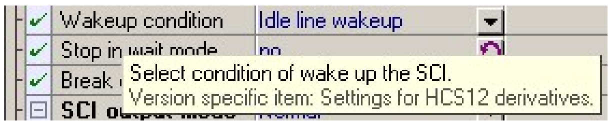
- use new value as default value — C:\ASM\
- [✓ use my default value] — D:\Program Files\PE 2.53 BETA
- use new value and do not change default value
- do not use default value for this item
- use allways default value for this item during project/template loading

Figure 2.28 - Changing the Default Value

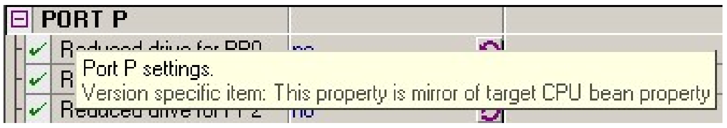
2.5.3.3. Version Specific Items

The **Version specific items** (properties, methods and events) are displayed only for CPU derivatives that support it. These items cover the special capabilities of the CPU and they are not present for all CPUs.

The version specific item is displayed as a highlighted field in the first column of the Bean Inspector. See the following picture. There are two items with this feature:


Wakeup condition
Idle line wakeup
Stop in wait mode
no
Break
SCI
Select condition of wake up the SCI.
Version specific item: Settings for HCS12 derivatives.

Some items of the bean are displayed as mirrored items from the CPU bean or global settings. The global setting means that it is possible to set the item in any bean where the item is available and the setting is used for all appropriate beans. The items are visible if they have any relation to the bean settings. The information about the mirroring is visible in the hint of the item.


PORT P

Port P settings.
Version specific item: This property is mirror of target CPU bean property

Figure 2.30 - Hint With Version Specific Info

2.5.4. Configuration Inspector

Configuration Inspector is a variant of an Inspector Window. It shows the settings that belong to one configuration. It could be invoked from configurations pop-up menu in the Project Panel (Click on a configuration with the right button and choose the *Configuration Inspector*). For details on configurations please see the chapter 3.3.2 Configurations.

Properties

The *Properties* tab contains optimization settings related to the configuration. These setting should be used when the code is already well debugged. They could increase speed of the code, but the generated code is less protected for the unexpected situations and finding errors could be more difficult.

- **Ignore range checking** - This option can disable generation of the code, that provides testing for parameter range. If the option is set to "yes", methods do not return error code ERR_VALUE neither ERR_RANGE. If the method is called with incorrect parameter, it may not work correctly.
- **Ignore enable test** - This option can disable generation of the code, that provides testing if the bean/peripheral is internally enabled or not. If the option is set to "yes", methods do not return error code ERR_DISABLED neither ERR_ENABLED. If the method is called in unsupported mode, it may not work correctly.
- **Ignore speed mode test** - This option can disable generation of the code, that provides a testing, if the bean is internally supported in the selected speed mode. If the option is set to "yes", methods do not return error code ERR_SPEED. If the method is called in the speed mode when the bean is not supported, it may not work correctly.
- **Use after reset values** - This option allows Processor Expert to use the values of peripheral registers which are declared by a chip manufacturer as the default after reset values. If the option is set to "no", all registers are initialized by a generated code, even if the value after reset is the same as the required initialization value. If the option is set to "yes", the register values same as the after reset values are not initialized.

Comment

The *Comment* tab allows to enter a text that will be shown in the hint when the mouse cursor will be placed on the configuration. It could be any text related to the configuration, for example an explanation of the configuration purpose or some necessary hardware settings.

2.6. Error Window

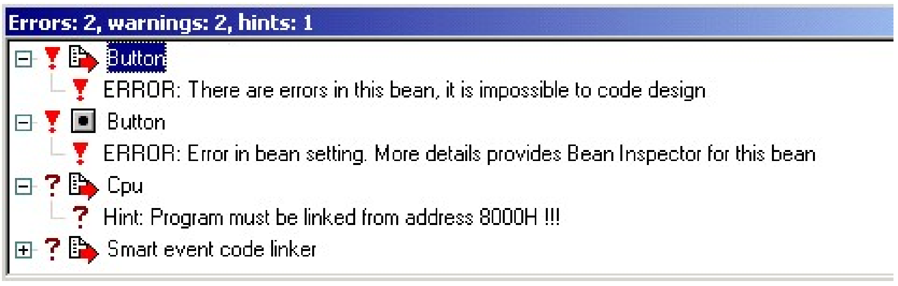
Processor Expert | View | Error window

This window displays errors, warnings, and hints that are found during:

- project checking
- code generation,
- running of an external tool.

Some errors are found right after inconsistent or incorrect data have been entered, others during the code generation of a project. The single messages mention the bean where the error was found. If an error concerns two beans (where conflict results for example from using the same on-chip peripheral), the error will be attributed to both beans.

If the user clicks the right mouse button a pop-up menu is shown allowing user to delete either tools or code generation errors, warnings and hints in order to improve the readability of the Error window.



Errors: 2, warnings: 2, hints: 1

Button
ERROR: There are errors in this bean, it is impossible to code design
Button
ERROR: Error in bean setting. More details provides Bean Inspector for this bean
Cpu
Hint: Program must be linked from address 8000H !!!
Smart event code linker

Figure 2.31 - Processor Expert Error window

Pop-up Menu

The pop-up menu invoked by a right mouse button click contains the following items:

- **Delete All Tool Errors, Warnings and Hits** - removes all tool errors, warnings and hints listed in the error window
- **Delete All code generation errors, warnings and Hints** - removes all code generation errors, warnings and hits listed in the Error window
- **Copy to Clipboard** - copies the whole content of the window as a text to the clipboard.
  Note: This command can be very useful in the case of contacting our support personnel with a bean setup issue.

- **Help** - display documentation

2.7. Target CPU Window

Processor Expert | View | Target CPU Package
Processor Expert | View | Target CPU Block Diagram
Processor Expert | View | Target CPU Structure

This window displays selected target CPU with its peripherals and pins (possible data directions of single pins are indicated by blue arrows on the CPU package when a bean uses these pins). Several **display modes** are supported. It is possible to switch the display mode by pushing buttons in the left side menu of the window.

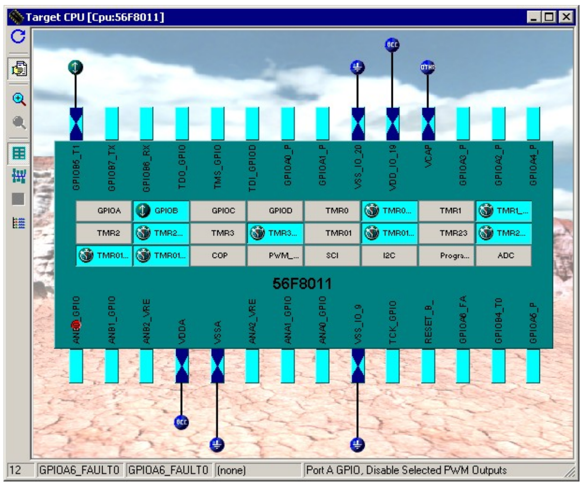
Control Buttons

The meanings of the buttons on the left side are:

- **Rotates CPU** - rotate the CPU 90 degrees to the right.
- **Show user names on CPU package** - switches the pins' and peripherals' default names (from catalog) for user-defined names.
- **Zoom in** - increases the detail level of the view. The whole picture might not fit the viewing area.
- **Zoom out** - decreases the detail level of the view. Processor Expert tries to fit the whole picture to the viewing area.
- **Show CPU package and peripheral** - switches to the CPU package view mode.
- **Show BGA CPU package** - switches to the CPU BGA package view mode.
- **Show CPU block diagram** - switches to the CPU block diagram view mode.
- **Show CPU peripherals in a list** - switches to the CPU peripherals list view mode.

View Modes

- **CPU package mode** - a realistic view of the CPU package with pins and peripherals. Each allocated peripheral contain an icon of the bean that allocates it. For allocated pins also the bean icon with the connection is shown.



12 | GPIOA6_FAULT0 | GPIOA6_FAULT0 | (none) | Port A GPIO, Disable Selected PWM Outputs

Figure 2.32 - Target CPU - CPU package view mode

- **CPU BGA package mode** - This mode is available only for CPUs with grid-array pins layout. It is similar to the package mode, but the pins hidden by package are shown and the peripherals are hidden.

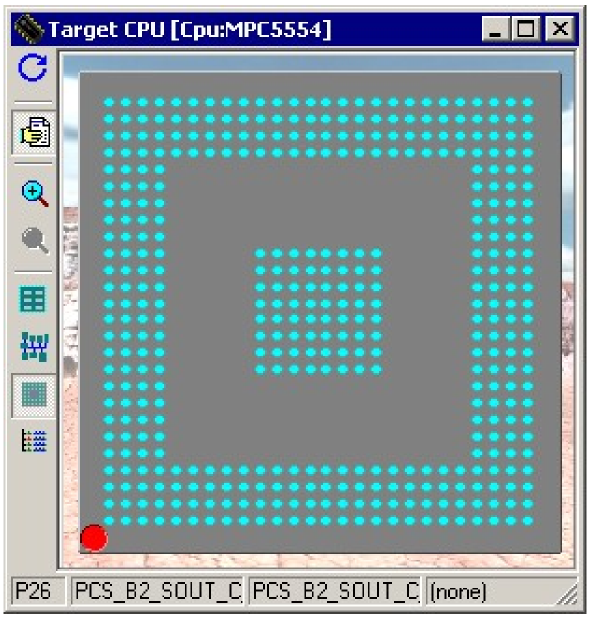


Figure 2.33 - Target CPU - BGA CPU package view mode

- **CPU block diagram mode** - a view of the CPU block diagram based on the documentation of the CPU manufacturer. Every part of the CPU is represented by a block. Every block that contains a resources that can be allocated by Processor Expert contains the slots for every resource (e.g. pin or channel). If the resource is allocated, the slot contains the icon of the allocating bean.

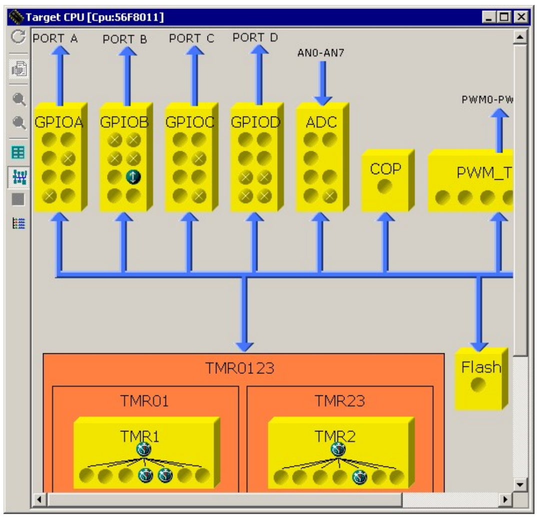


Figure 2.34 - Target CPU - MCU block diagram view mode

- **CPU peripherals list** - a list of all peripherals of the CPU is displayed. If a peripheral is unallocated by Processor Expert, it is displayed as a gray icon. Otherwise, the icon of the bean that allocates the peripheral is displayed. The same mouse commands are available as in the other view-modes, except the operations with pins (pins are not visible in this mode).

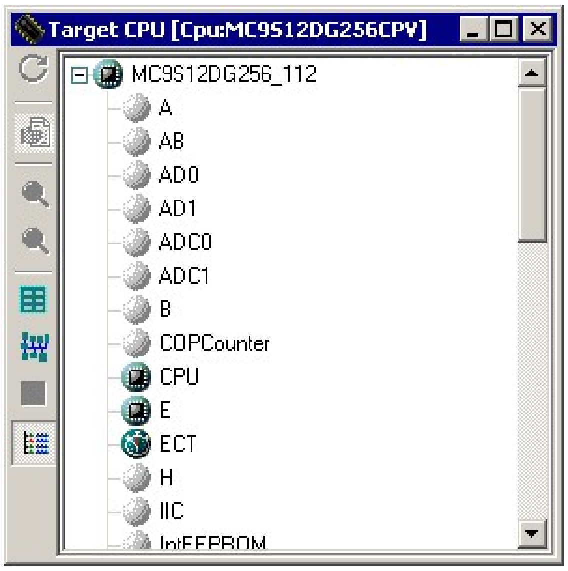


Figure 2.35 - Target CPU - Peripherals List Mode

Pins

The following information about each pin is displayed on the CPU picture:
(all pins are displayed only in the CPU package view mode)

- pin name (default or user-defined)
- icon of a bean that uses (allocates) the pin
- direction of the pin (input, output, or input/output) symbolized by blue arrows, if a bean is connected

Pin names are shortened and written either from left to right or from up to down and are visible only if there is enough space in the diagram.

Some signals and peripherals cannot be used by the user because they are allocated by special devices such as power signals, external or data bus. The special devices are indicated by a special blue icons, for example ⏚. The allocation of peripherals by special devices can be influenced by CPU properties.

Hints

Pin hint contains:

- number of the pin (on package)
- both names (default and user-defined)
- owner of the pin (bean that allocates it)
- short pin description from CPU database

Bean icon hint contains:

- bean name
- bean type
- bean description

Shared Pins

If a pin is shared by multiple beans, the line connecting the pin to the bean has a red color. See chapter *3.3.8 Pin*

Sharing for details.

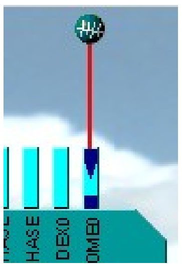


Figure 2.36 - Shared pin connection

On-chip peripherals

The following information about each on-chip peripheral is displayed on the CPU package:

- peripheral device name (default or user-defined)
- icon of the bean that uses (allocates) the peripheral device

Peripheral device hint contains:

- peripheral device name
- owner of the pin (bean that allocates it)
- short peripheral device description

Hint on icon contains:

- bean name
- bean type
- bean description

If a peripheral is shared by several beans (for example: several beans may use single pins of the same port), the icon is displayed.

Note for peripherals working in several modes:
Some peripherals work in several modes and these peripherals can be represented by a several devices in the CPU databases. For example, the device "TimerX_PPG" and "TimerX_PWM" represents TimerX in PPG and in PWM mode. These devices can be displayed on the CPU package, but they are also represented as a single block in the MCU block diagram.

Mouse Operations For Individual Items

- Single click on a bean icon selects the bean in the Project panel.
- Double click on a **bean icon** opens its Bean Inspector and selects the property specifying the peripheral used by the bean.
- Double click on a **peripheral** opens the simple item structure view.
- Double click on an icon opens a selection menu with all the beans that use single parts of the peripheral. Selecting one bean opens it in the Bean Inspector.
- Right button click on a **bean icon** opens the Bean pop-up menu. If the Bean Inspector is invoked from this pop-up menu, an appropriate property allocating the used peripheral is selected.
- Right button click on an icon opens selection menu with all the beans that use single parts of the

peripheral. Selecting one bean opens the Bean pop-up menu.

- Right click on the peripheral opens the Peripheral Pop-up menu (see below).

Peripheral/Pin Pop-up Menu

The following commands are available in the pop-up menu:

- **Show Peripheral Initialization** - shows initialization values of all "control, status and data" registers. This option is supported for all devices displayed on a CPU package. See chapter *2.14 Peripheral Initialization* for details.
- **Show Peripheral Structure** - opens the peripheral's structure view - (it is supported for I/O ports, timer's counters, serial ports. This option is also supported for devices working in several modes in the CPU block diagram. A list of represented devices for these modes is displayed.
- **Show Peripheral Usage** - shows which part of the peripheral is used by the application (visible after code generation). This option is supported for I/O ports and pins, timers, A/D converters and A/D channels, CAN, serial ports, watchdog, internal memories (EEPROM and FLASH). See chapter *2.15 Peripherals Usage* for details.
- **Rename Peripheral** - allows you to rename the selected peripheral. It is supported for I/O ports and pins, watchdog and timers (counters, compare and capture registers, free running devices), A/D converters and A/D channels, CAN, serial ports.
- **Search Related Info In CPU PDF Documentation** displays PDF Search window and finds the information about the peripheral in the appropriate CPU documentation. It is also possible to search for any keyword in the CPU documentation based on the original manufacturer's CPU manual. (This item is available on the package and on the CPU block only.) See chapter *2.17 PDF Search* for details.
- **Add Bean/Template** - adds a bean or template for the appropriate peripheral: all available beans and templates suitable for the selected peripheral are listed. The beans and templates in the list are divided by a horizontal line. It is possible to add only beans or templates which are applicable for the peripheral. It means that is possible to add the bean or template only if the peripheral is not already allocated to another bean or beans. The beans/templates that cannot be added to the peripheral are grayed in the pop-up menu as unavailable. This option is supported for all devices displayed on CPU package.
- **Help on Target CPU Window** - displays help for the current window

2.8. CPU Timing Model

Project Panel | CPU pop-up menu | View CPU Timing Model

This window presents the schematic structure of the target CPU timing stored in the database of Processor Expert. The information can be displayed in two levels of complexity:

- **Simple view** - In this mode, only the active prescalers are shown. Unimportant tree nodes like inactive branches, divider by one etc... are hidden. Every clock source is shown as a node of the tree at the topmost level. *This is the default mode.*
- **Normal view** - All nodes of the timing model are shown.

The complexity can be switched by checking/un-checking the *Simple View* item of the pop-up menu of the window.

Click plus "+" or minus "-" signs to expand or collapse the branches of the tree.

Icons Description

- ✓ - active endpoint - a device using the clock.
- ✗ - inactive device or unused branch
- ⏚ - clock source
- % - adjustable prescaler
- % - constant prescaler
- ✖ - adjustable multiplier
- ✖ - constant multiplier
- ⊣ - clock distribution to multiple branches
- ≺ - element selecting one of the connected branches. The inactive branches have the gray cross (✗) icons (for an inactive branch example please see the **RTIJoin** prescaler on the figure below).
- ≻ - join of multiple clock sources

The values of all prescalers are for high speed mode. For details on speed modes please see the chapter 3.2.2.2 Speed Modes Support.

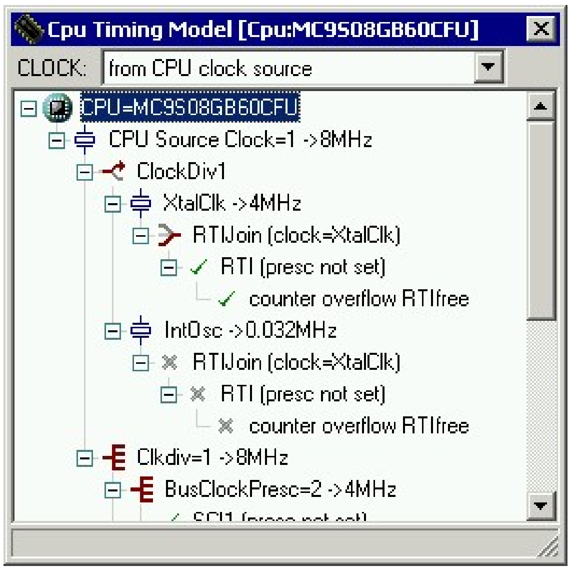
More details on individual items are available as hints after the mouse cursor is placed on the item's name. The displayed prescaler values are automatically set after the reset for enabled devices. The values are influenced by the beans configuration. For details on the bean timing principles please see the chapter 3.3.4 Timing Settings.

Some nodes contain a frequency information. It is a clock frequency on the node output.

Clock Path Direction

It is possible to change the direction of the timing path. This can be set by switching the CLOCK field value:

- **From CPU clock source to individual peripherals**
  The prescalers and clock-driven CPU devices are ordered in a tree structure, starting with the main branch which represents the main clock source (PLL, X-tal etc...).



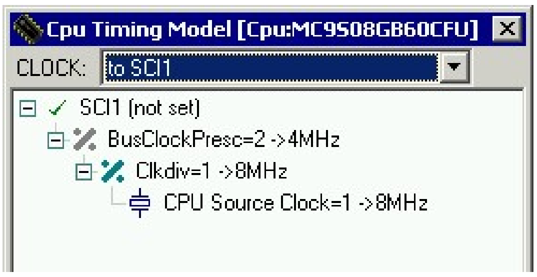
Cpu Timing Model [Cpu:MC9S08GB60CFU]

CLOCK: from CPU clock source

```
CPU=MC9S08GB60CFU
  CPU Source Clock=1 ->8MHz
    ClockDiv1
      XtalClk ->4MHz
        RTIJoin (clock=XtalClk)
          RTI (presc not set)
            counter overflow RTIfree
      IntOsc ->0.032MHz
        RTIJoin (clock=XtalClk)
          RTI (presc not set)
            counter overflow RTIfree
    Clkdiv=1 ->8MHz
      BusClockPresc=2 ->4MHz
        SCI1 (presc not set)
```

Figure 2.37 - CPU Timing Model Window

- **From selected peripheral to CPU clock source**
  The selected peripheral is the root of a tree showing the current sequence of the prescalers form the device to the main clock source. Each sub-node represents a source of the clock used for its parent.



```
SCI1 (not set)
  BusClockPresc=2 ->4MHz
    Clkdiv=1 ->8MHz
      CPU Source Clock=1 ->8MHz
```

Figure 2.38 - CPU Timing Of Selected Peripheral

2.9. Resource Meter

Processor Expert | View | Resource Meter

The Resource Meter shows the current status of a chip resource usage (or availability).
Note that if a peripheral is allocated, all its parts are reserved. For example if you use the 8-bit I/O port, all the I/O pins of the port are allocated and it is not possible to use them in other beans.

- **Pins usage** meter shows pins' usage. In general, there are always some pins used, such as the power supply pin.
- **Port usage** meter shows ports' usage. A port is considered allocated if a part of it is used, or if its pins are allocated to another device.
- **Compare/Reload** meter shows the allocation of the timer compare registers (depending on CPU type). If it is possible to combine several smaller registers into one large, then the allocation of one of the smaller ones means allocation of the large one and the allocation of the large one means allocation of the two smaller ones.
- **Capture regs** shows timer capture registers usage.
- **Communication** shows the allocation of the serial communication channels (including also CAN).
- **A/D channels** shows the allocation of the A/D converter channels.

By placing the mouse over a resource meter field, you may get a hint that provides details about which resources are used concerning this field.

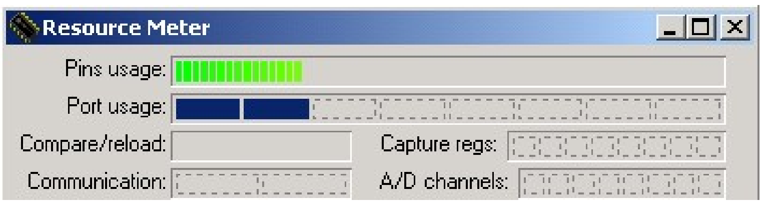


Figure 2.39 - Resource Meter Window

2.10. Memory Map Window

Processor Expert | View | Memory Map

This window shows the CPU address space and **internal** and **external memory** mapping. Detailed information for an individual memory area is provided as a hint when the user moves cursor over it.

Legend:

⬜	white: non-usable space
🟦	dark blue: I/O space
🟦	blue: RAM
🟦	light blue: ROM, OTP or Firmware
🟩	cyan: FLASH memory or EEPROM
⬛	black: external memory

The address in the diagram is increasing upwards. The sizes of individual memory areas blocks drawn in the window are different from the ratio of their real sizes to improve readability of the information (Small blocks are larger and large blocks are smaller).

The black line-crossed areas show the memory allocated by a bean or compiler. The address axis within one memory block goes from the left side to the right (i.e. the left side means start of the block, the right side means the end).

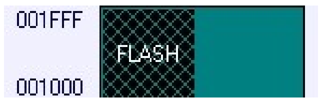


Figure 2.40 - Sample Of Used Part Of The Memory Area

Mouse Actions

If you move the mouse pointer to any part of the CPU address space, a detailed description of the chosen part will be displayed in the hint.

Double click on the used (line-crossed) memory area will open the CPU bean inspector window with the selected definition for this area (same as the Edit Usage pop-up menu command).

Pop-up Menu

- **Edit Usage** - opens a Bean Inspector window that allows to customize the memory setup for the selected area. This command is available only for used areas (line-crossed).
- **Display Mode**
  - **Full** - displays all items.
  - **Only Memory** - displays only Flash, RAM and ROM.
- **Summary** - displays window with summary of memory usage (percentage and absolute view).
- **Help** - shows this page.

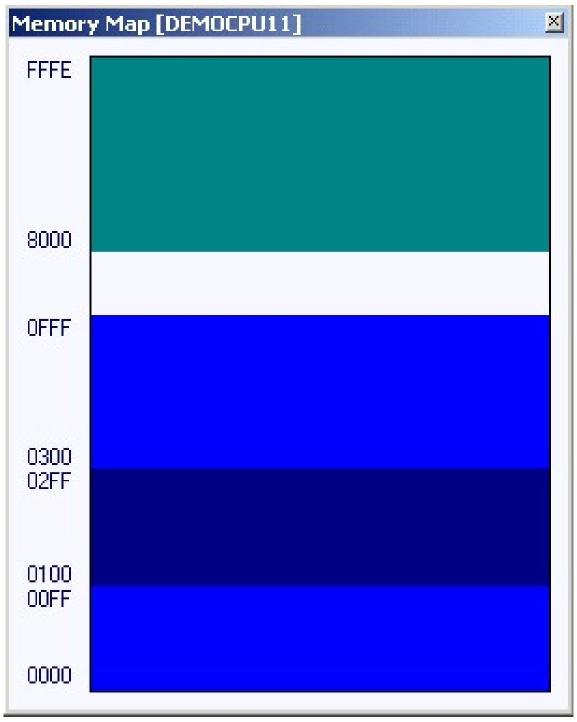


Figure 2.41 - Sample Memory Map Window

2.11. CPU Types Overview

Processor Expert | View | CPU Types Overview

The **CPU Types Overview** dialog window provides an overview of all CPUs available in user's installation of Processor Expert. The overview is given in the form of a tree structure going from CPU producers to CPU families and finally CPU variants. The schematic picture of the currently selected CPU variant is displayed in the Target CPU window.

- A mouse click on the **close button** will close the window.
- A mouse click on the **help button** will open this help page.

If you click with the right mouse button on the window, a pop-up menu appears:

- **Add CPU into project** - adds the selected CPU into the project (it will appear in the Project Panel).
- **Expand/Collapse** - expands/collapses the next level of the tree structure or packs the current one.

- **Expand all** - expands the entire tree structure.
- **Collapse all** - expands the entire tree structure.
- **Help** - opens the help page concerning this window.

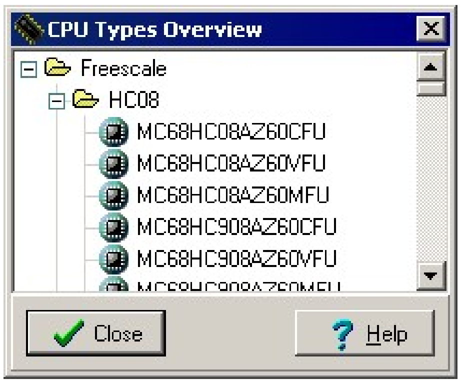
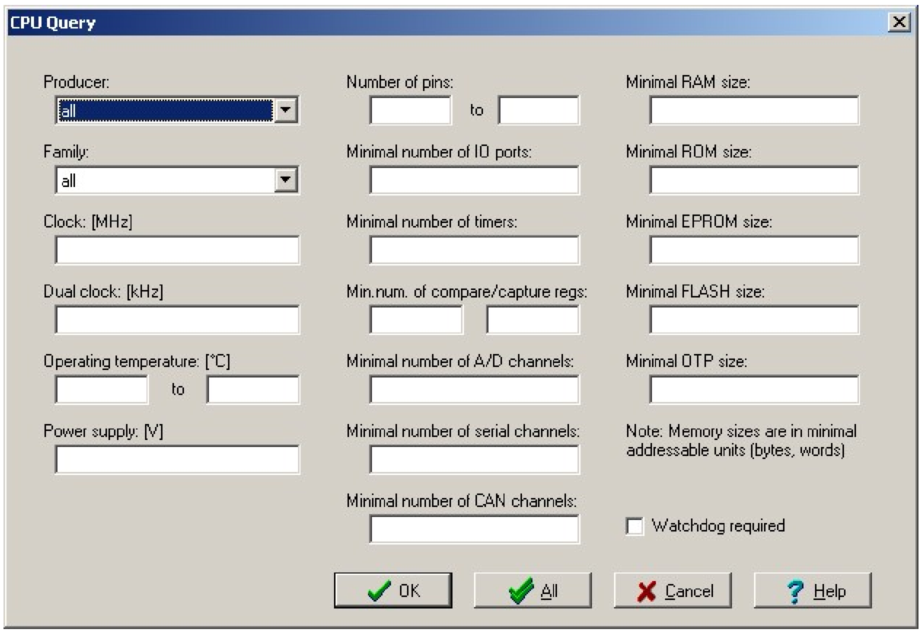


Figure 2.42 - CPU Types Window

2.12. CPU Parameters Overview

Processor Expert | View | CPU Parameters Overview

You may get the technical features of a CPU by selecting the **CPU Parameters Overview** command of the View menu. The complete database then appears together with a query window. By specifying requirements on technical features, you may filter the database in order to display only relevant CPUs. If you press **OK**, you will get the list of CPUs that meet your requirements. If you press **All**, all the CPUs supported by your version of Processor Expert will be listed.


CPU Query

Producer: all
Family: all
Clock: [MHz]
Dual clock: [kHz]
Operating temperature: [°C] to
Power supply: [V]
Number of pins: to
Minimal number of IO ports:
Minimal number of timers:
Min.num. of compare/capture regs:
Minimal number of A/D channels:
Minimal number of serial channels:
Minimal number of CAN channels:
Minimal RAM size:
Minimal ROM size:
Minimal EPROM size:
Minimal FLASH size:
Minimal OTP size:
Note: Memory sizes are in minimal addressable units (bytes, words)
☐ Watchdog required

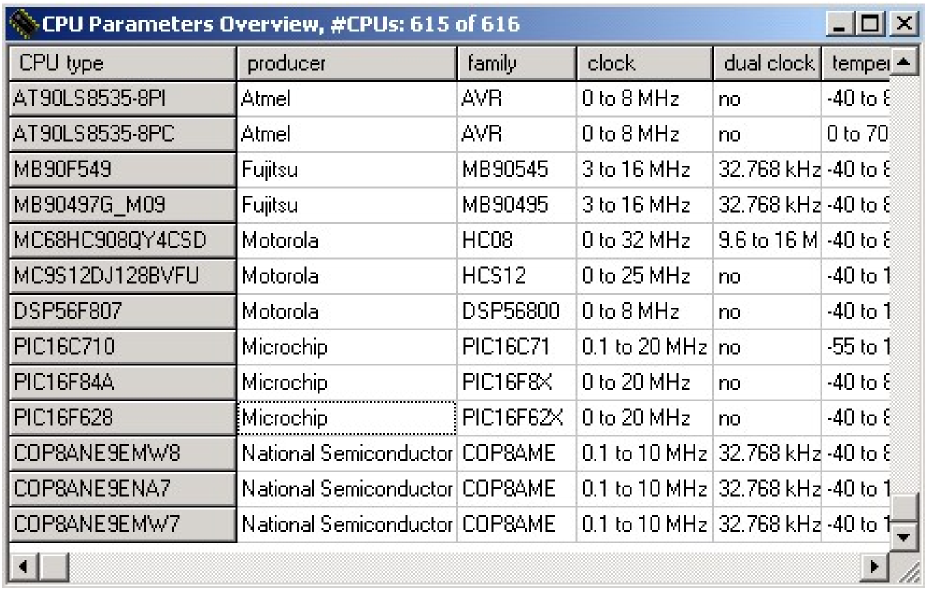
Figure 2.43 - CPU Query Dialog

Description

The **CPU Parameters Overview** windows displays the list of CPUs, including their technical features:

- *CPU type* - CPU type
- *Producer* - CPU producer
- *Family* - CPU family
- *Clock* - CPU xtal clock
- *Dual clock* - if CPU does have a dual clock
- *Operating Temperature* - CPU operating temperature
- *#pins* - number of pins on CPU package
- *#IO ports* - number of I/O ports and I/O pins
- *#timers* - number of timers, compares and captures
- *#A/D* - number of A/D channels and converters. For example 12/1 means 12 A/D channels and 1 A/D converter.
- *#serial* - number of asynchro/synchro serial channels
- *#CAN* - on-chip CAN channels
- *Watchdog* - on-chip watchdog
- *#Special* - Special features and devices
- *RAM* - on-chip RAM size
- *ROM* - on-chip ROM size
- *EPROM* - on-chip EPROM size
- *FLASH* - on-chip FLASH size
- *OTP* - on-chip OTP size
- *Power supply* - power supply voltage
- *Storage* - CPU storage temperature

Note: Memory sizes are in minimal addressable units (bytes, words).
If you right-click on the window, a menu appears allowing you to add the selected CPU to the current project or to refine your previous query.


CPU Parameters Overview, #CPUs: 615 of 616

CPU type	producer	family	clock	dual clock	tempe
AT90LS8535-8PI	Atmel	AVR	0 to 8 MHz	no	-40 to 8
AT90LS8535-8PC	Atmel	AVR	0 to 8 MHz	no	0 to 70
MB90F549	Fujitsu	MB90545	3 to 16 MHz	32.768 kHz	-40 to 8
MB90497G_M09	Fujitsu	MB90495	3 to 16 MHz	32.768 kHz	-40 to 8
MC68HC908QY4CSD	Motorola	HC08	0 to 32 MHz	9.6 to 16 M	-40 to 8
MC9S12DJ128BVFU	Motorola	HCS12	0 to 25 MHz	no	-40 to 1
DSP56F807	Motorola	DSP56800	0 to 8 MHz	no	-40 to 1
PIC16C710	Microchip	PIC16C71	0.1 to 20 MHz	no	-55 to 1
PIC16F84A	Microchip	PIC16F8X	0 to 20 MHz	no	-40 to 8
PIC16F628	Microchip	PIC16F62X	0 to 20 MHz	no	-40 to 8
COP8ANE9EMW8	National Semiconductor	COP8AME	0.1 to 10 MHz	32.768 kHz	-40 to 8
COP8ANE9ENA7	National Semiconductor	COP8AME	0.1 to 10 MHz	32.768 kHz	-40 to 1
COP8ANE9EMW7	National Semiconductor	COP8AME	0.1 to 10 MHz	32.768 kHz	-40 to 1

Figure 2.44 - Results of The CPU Query

2.13. List of Installed Beans with Additional Information

Processor Expert | View | Installed Beans Overview

This window contains information about **installed beans** in the current version of Processor Expert:

- **BEANS**
  All installed beans including CPUs and user-created beans are listed in the report. Placing the mouse cursor on the bean's name will display the hint containing the detailed information about the latest bean version.
  See **menu | View | Bean Type** for bean type selection.
- **Bean Info**
  The amount of the information shown in this column can be controlled via **View | Bean info** menu.

  Note: This information is also displayed as a hint on the bean.

  - Bean description
  - File format and access *(encrypted, full source, compressed)*
  - Author of the bean
  - version
  - List of revisions of the bean. Each revision contains a version number, date and author of the revision.
- **Drivers**
  Installed bean's drivers.
- **Driver Info**
  The amount of the information shown in this column can be controlled via **View | Driver info** menu.

  Placing the mouse cursor on the file name will display the detailed information about the latest bean driver version.

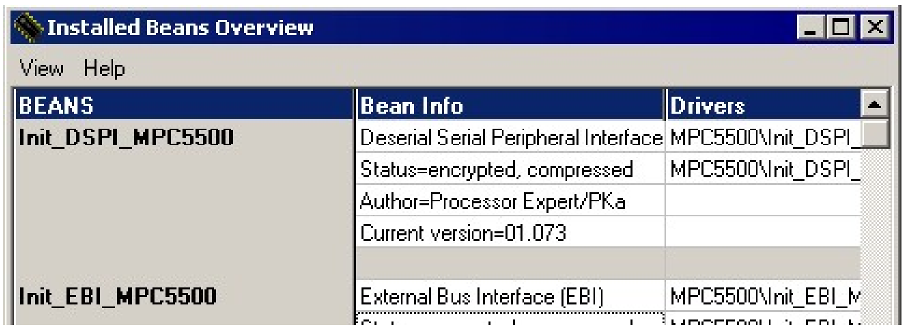
  - Status - file format and access *(encrypted, full source, compressed)*
  - Author of the bean's driver

- Current version of the bean's driver
- Init. date - date of creation
- Last modification - date of last modification
- List of revisions of the driver. Each revision contains a version number, date and author of the revision.

The contents of the window can be configured using commands in menu **View**. It is possible to view this window also in HTML format using command **View | As HTML**.

Menu

- **View**
  - **Bean Type**
    - **all** - displays all beans.
    - **Beans only** - hides CPU beans.
    - **CPUs only** - displays CPU beans only.
  - **Bean Info**
    - **all** - displays information about the bean.
    - **basic** - displays basic information about the bean only.
    - **none** - hides information about the bean.
  - **Drivers**
    - **all** - displays all bean's drivers.
    - **none** - hides all drivers.
    - **subdirectories** - displays drivers from a selected subdirectory.
  - **Driver Info**
    - **all** - displays the most information about bean's drivers.
    - **basic** - displays only a basic information about the bean's driver.
    - **none** - hides information about bean's drivers.
  - **Implementation**
    - **display all** - shows all beans.
    - **Hide beans without any selected driver** - shows only beans which contain the selected driver.
    - **Hide beans with any selected driver** - shows only beans which do not contain any selected driver.
  - **As HTML** - display this table in default HTML browser.
    Temporary HTML file will be removed from the disk after closing Processor Expert.
- **Help**
  - **Help on Bean**- displays help for currently selected bean (except CPU beans).
  - **Help on this Window** - displays this page.



BEANS	Bean Info	Drivers
Init_DSPI_MPC5500	Deserial Serial Peripheral Interface	MPC5500\Init_DSPI_
	Status=encrypted, compressed	MPC5500\Init_DSPI_
	Author=Processor Expert/PKa	
	Current version=01.073	
Init_EBI_MPC5500	External Bus Interface (EBI)	MPC5500\Init_EBI_M

Figure 2.45 - List of Installed Beans

2.14. Peripheral Initialization

Processor Expert | View | Peripheral Initialization

The Peripheral Initialization window shows overview of peripheral initialization settings for the current CPU. It displays initialization values of all *control, status and data* registers of selected peripheral including single bits. The user can also see peripheral schematic diagram. Peripheral Initialization can be invoked from the View menu or from Target CPU window or using a Peripheral Initialization button in the Bean Inspector.

If the Peripheral Initialization window is docked with the Bean Inspector window, the peripheral is automatically changed according to the peripheral selection in the bean selector. Automatic changes of the peripheral can be disabled using the lock icon .

Notice: The Peripheral Initialization and Peripheral Usage are both only one window in two different modes. These windows could not be displayed both at once.
The initialization information reflects:

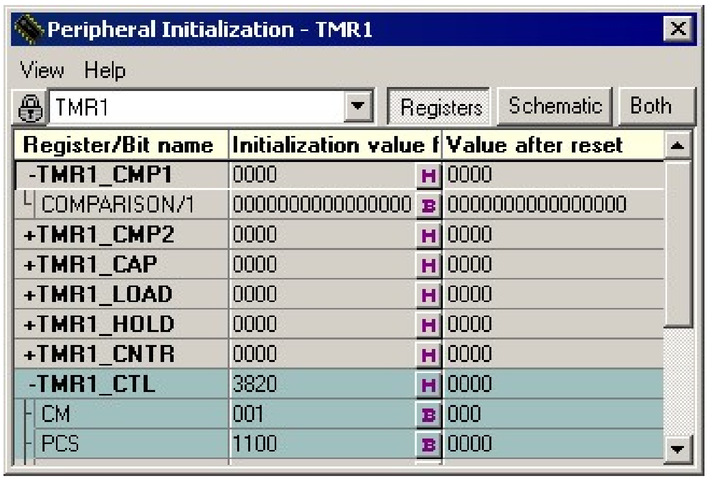
- CPU default settings - when the peripheral is not utilized by Embedded Bean
- Embedded Bean settings - when the peripheral is utilized by the Embedded Bean and the bean settings are correct. Peripheral Initialization Inspector shows initialization as required by Beans settings.

Registers

There is a value displayed in the middle column which will be written into the register or bit by the generated code during the initialization process of the application. It is the last value that will be written by the initialization function to the register.

Note: For some registers, the value read from the register afterwards can be different than the last written value. For example, some interrupt flags are cleared by writing 1. Please see the CPU manual for details on registers behavior.

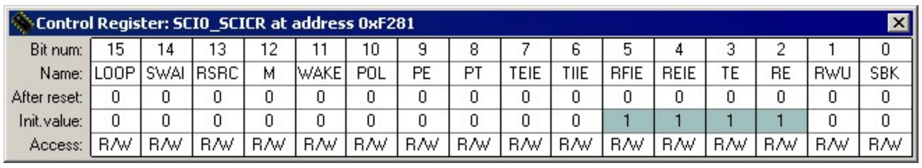
In case the peripheral is allocated by a bean and the setting of the bean is incorrect, then the initialization values are not displayed in the Peripheral Initialization window. Instead, a value of the register (or bit) after reset is displayed in the right column. The *after-reset values* can contain also a characters with special meaning. The list and description of these characters is displayed as a hint when the mouse cursor is placed on the header of the registers table.
The values of the registers can be displayed in hexadecimal, decimal or in binary form. In case the value of the register (or bit) is not defined, an interrogation mark "?" is displayed instead of the value. In this case it is possible to display the value of the register in binary form only.



Register/Bit name	Initialization value f		Value after reset
-TMR1_CMP1	0000	H	0000
COMPARISON/1	0000000000000000	B	0000000000000000
+TMR1_CMP2	0000	H	0000
+TMR1_CAP	0000	H	0000
+TMR1_LOAD	0000	H	0000
+TMR1_HOLD	0000	H	0000
+TMR1_CNTR	0000	H	0000
-TMR1_CTL	3820	H	0000
CM	001	B	000
PCS	1100	B	0000

Figure 2.46 - Register List in the Peripheral Initialization window

Register details command from pop-up menu of a register (invoked by the right button click) opens a window containg detailed information for the register.



Bit num:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name:	LOOP	SWAI	RSRC	M	WAKE	POL	PE	PT	TEIE	TIIE	RFIE	REIE	TE	RE	RWU	SBK
After reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Init.value:	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0
Access:	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Figure 2.47 - Register Content Details

Changes Highlighting

The user can watch reflections of user settings to Embedded Bean Properties directly in CPU peripheral registers and bits. The registers influenced by a last bean settings change are highlighted with a green color (see the previous two pictures for example). The highlighting works only in the case that the bean was set-up correctly before the change was made and the new setup is correct as well and there is no error reported in a bean settings. If there is an error in a bean settings and no other bean is influencing the register, the after-reset values are are shown.

Menu

- **View**
  - **Sort Registers by Address** - If this item is checked, registers are sorted in list by their address. Otherwise they are sorted by name.
  - **Group Registers** - If this item is checked, groups of numbered registers with the same name are shown as expandable folders. The name of each folder is the same as the name of the registers with numbers replaced by 'xx'.
  - **Show Unused Bits** - This option enables/disables displaying the registers' bits unused by the manufacturer. When it is enabled, the unused bits are listed with the name 'unused'.
  - **Expand all** - expands the folder and all its subfolders.
  - **Collapse all** - collapses the folder and all its subfolders.
- **Help**

- **Help** - displays documentation.

Peripheral schematic

It is possible to switch peripheral initialization window style between register list, peripheral schematic and both of them using **registers**, **schematic** and **both** buttons. *Schematic and both buttons are present only when the schematic for the current peripheral is provided.*

Schematic diagram of the peripheral contains names of the control registers. Highlighting of changes of a related bean settings works also in the schematic mode.

- If the user places mouse cursor on the register name, the hint containing register description and address is shown.
- When the user clicks the register name with the left mouse button, new dialog window containing **detailed information** (including initialization value) for the register is shown. The register bits affected by a changes are highlighted.

Notice: Peripheral schematics view is available only in the 56F800/E version.

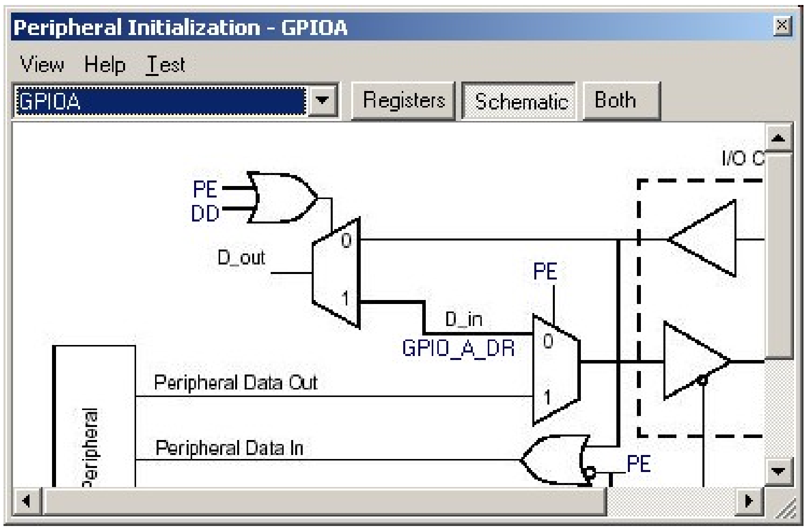


Figure 2.48 - Schematic Diagram of the Peripheral

2.15. Peripherals Usage

Processor Expert | View | Peripherals Usage

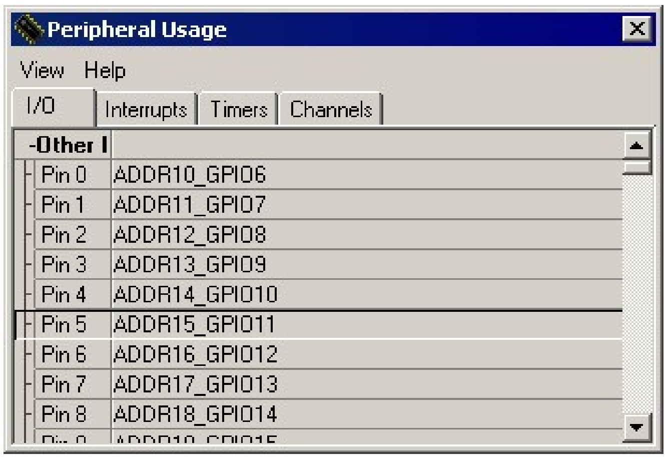
The Peripheral Usage window shows the current status of on-chip peripherals usage in detail.
The names of the beans using peripherals appear in the third column, next to the corresponding peripheral names. When the mouse is placed over a peripheral name, a short description of the peripheral appears as a hint.
Note that if a peripheral is allocated, all its parts are reserved. For example if you use the 8-bit I/O port, all the I/O pins of the port are allocated and it is not possible to use them in other beans.
The list can be filtered for only the used peripherals/interrupts/channels to be shown by using the menu command **View | Show Used Peripherals Only**.

Notice: The Peripheral Initialization and Peripheral Usage are both only one window in two different modes. These windows could not be displayed both at once.
The following items are available:

- **I/O page** shows I/O pins' and ports' usage. If a pin/port is allocated to a bean, the I/O properties of the bean are displayed below the pin/port's name (unless all the pins of a port are used by one bean, in which case, the properties appear under the name of the allocated pins).
- **Interrupts page** shows interrupt vectors usage. Each item of the list contains a number of the interrupt vector and its name. If an interrupt vector is allocated to a bean, the bean's name and the interrupt priorities are displayed as a sub-items in the vector's group.
- **Timers page** shows the allocation of the timers.
- **Channels page** shows A/D, CAN and serial channels usage.

Menu

- **View**
  - **Sort registers by address**, **Group registers**, **Show Unused Bits** - In this mode of the window these commands have no use and are not available. **Show Used Peripherals Only** - After this option is enabled, only the peripherals allocated by Processor Expert are shown.
  - **Expand all** - expands the folder and all its subfolders.
  - **Collapse all** - collapses the folder and all its subfolders.
- **Help**
  - **Help** - displays documentation.



Pin 0	ADDR10_GPIO6
Pin 1	ADDR11_GPIO7
Pin 2	ADDR12_GPIO8
Pin 3	ADDR13_GPIO9
Pin 4	ADDR14_GPIO10
Pin 5	ADDR15_GPIO11
Pin 6	ADDR16_GPIO12
Pin 7	ADDR17_GPIO13
Pin 8	ADDR18_GPIO14

Figure 2.49 - Peripheral Usage Window

2.16. File Editor

About File Editor

File editor is the Processor Expert internal editor allowing to

- Edit files - All common text editor functions are available for comfortable work with the source code.
- View files - Editor is opened in a read only mode (it's shown in the title of the window).
- Compare files - A visual file-comparison mode with two panels showing a differences between two files.

Notice: The Edit and View modes of the internal editor are not used in the CodeWarrior plugin. Internal editor is for these operations fully replaced by the native CodeWarrior source code editor.

Meaning of Buttons:

- **Switches between module Extensions** - switch between assembler implementation of the driver and header file of the module or implementation (body file) of the module.
  Note: it is available only if it is allowed in the Editor Options.
- **Open file** - opens a file.
- **Save file** - saves the currently displayed file.
- **Save all** - saves all opened files (if they were edited).
- **Close file** - closes the currently displayed module (file).
- **Print file** - prints the currently displayed file on a printer.
- **Editor options** - editor settings.
- **Help** - opens Help.
- **Change font** - changes the font of File Editor.
- **Undo** - restores the state of a file before the last change.
- **Redo** - restores the last change.
- **Find** - finds a string in the currently displayed file.
- **Replace** - replaces a string with another string.

Mouse Actions

- To move editor cursor to a specific place, click the left button on the desired place in a text.
- To select a text, move mouse and hold the left button.
- To move the selected text to a different location, drag a selected text with left button pressed.
- To invoke editor pop-up menu, click the right button.
- To scroll editor view, move mouse with middle button pressed.
- To close an opened file, click the middle button on the tab with a file name (when multiple files are open).
- To select a square text block inside the window press and hold the ALT key and left mouse button while moving the mouse.
- To jump on a specified line number, click the first status-bar field containing a line number information.
- To place a method invocation to the source code, drag the method from the Project Panel. See chapter *2.3 Project Panel* for details.

File Editor Pop-up Menu

To open the File Editor pop-up menu, click the right mouse button on the text area of the File Editor window.

Meanings of items:

- **File**

- **Open** - opens a file.
- **Save** - saves the currently displayed file.
- **Save As** - saves the currently displayed file under a new name.
- **Save All** - saves all opened files (if files were edited.
- **Close**- closes the currently displayed file.
- **Close all** - closes all files.
- **Delete** - closes and deletes the currently displayed file.
- **Reopen** - opens a list of used files.
- **Print** - prints the currently displayed file on a printer.
- **Print Preview ...** - opens a print preview dialog. The dialog allows the user to enter printer options, set zoom size for preview, display document margins and print the page.
- **Edit**
  - **Undo** - Restores the state of a file before the last change.
  - **Redo** - Resumes the last change.
  - **Cut** - Cuts the selected text to the clipboard.
  - **Copy** - Copies the selected text to the clipboard.
  - **Paste** - Pastes the text from the clipboard.
  - **Clear** - Clears the selected text.
  - **Select All** - Selects all text.
  - **Delete Line** - Removes the current line.
  - **Delete word** - Removes characters from the current cursor position to the end of the word (including space).

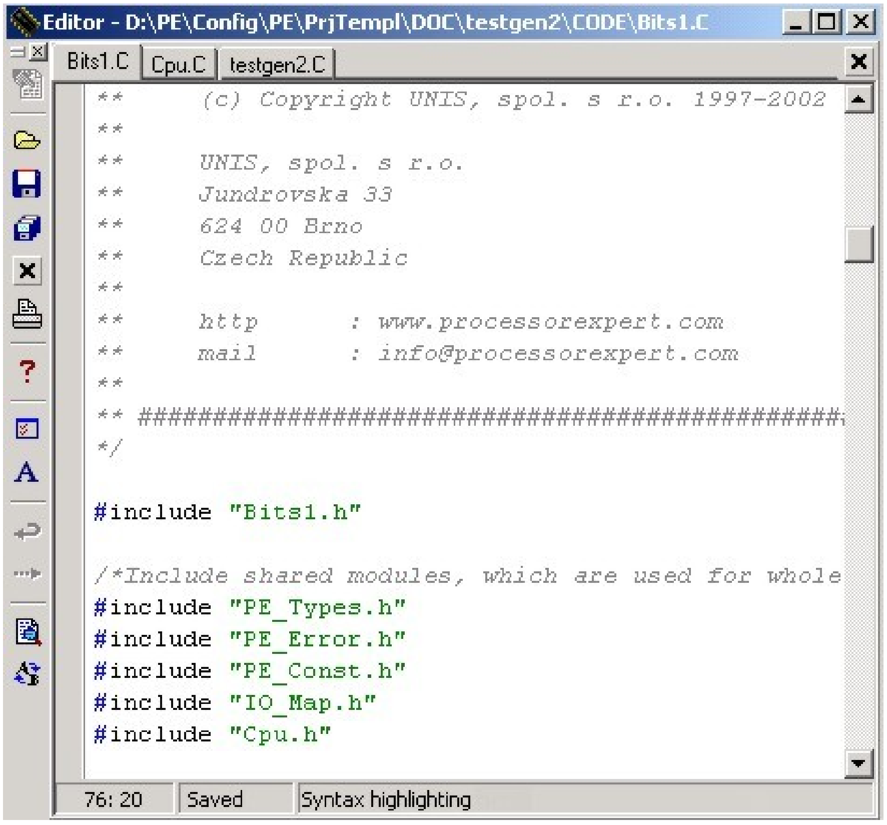
- **Search**
  - **Find** - Displays SearchReplace dialog
  - **Find Next** - Finds next position of a string in the currently displayed file.
  - **Replace** - Displays SearchReplace dialog
  - **Go to Line** - Moves the cursor to a line determined by its number.
- **Debug**

Notice: Internal debugger is not available in this version.

- **Toggle break** - Set breakpoints on actual cursor position.
- **Add watch ...** - Add watch to watches list in Watches window.
- **Evaluate/Modify ...** - Open watch editor and enable to edit value of any variable.
- **Run** - Runs application in target system.
- **Go to cursor**- Runs application and it stops on actually selected row and breakpoint.
- **Stop** - Stops running application.
- **Reset** - Resets application in target system.
- **Step into** - Steps program including subprogram call.
- **Step over** - Steps program without subprogram call.
- **Step into in assembler** - Steps program including subprogram call. After one assembler instruction execution actual program position is displayed in window.
- **Step over in assembler** - Steps program without subprogram call. After one assembler instruction execution actual program position is displayed in window.
- **Go to Source Line** - Opens window and goes to actual program position.
- **Modules** - Shows modules list into ABS file.
- **Disassembler** - Opens Disassembler for the current module.
- **Disassembler Whole Code** - Opens Disassembler for the whole program memory.
- **Options**
  - **Editor Options...** - setting of the File Editor

  (see Editor Options below).
  - **Change Font...** - change font of File Editor.
  - **Disassembler Op-codes**

  These commands in this submenu are not supported in this version of Processor Expert.
- **Editor Toolbar Visible** - show or hide toolbar of the File Editor. The toolbar is visible as a part of the window or as a single panel. Clicking on the toolbar corner it is possible to place the toolbar to other side of the window or to any other place on the screen as a floating panel.
- **Help** - opens documentation.



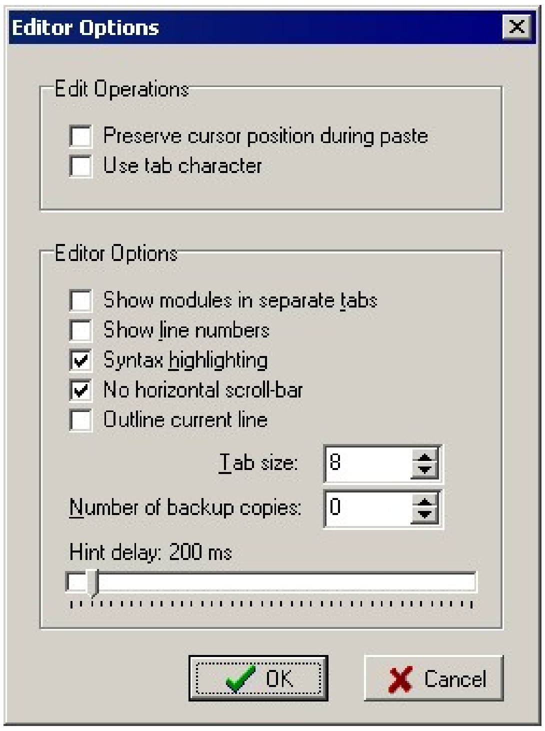
```
**      (c) Copyright UNIS, spol. s r.o. 1997-2002
**
**      UNIS, spol. s r.o.
**      Jundrovska 33
**      624 00 Brno
**      Czech Republic
**
**      http      : www.processorexpert.com
**      mail      : info@processorexpert.com
**
** ##################################################
*/

#include "Bits1.h"

/*Include shared modules, which are used for whole
#include "PE_Types.h"
#include "PE_Error.h"
#include "PE_Const.h"
#include "IO_Map.h"
#include "Cpu.h"
```

76: 20 | Saved | Syntax highlighting

Figure 2.50 - File Editor

File Editor Options

- **Preserve cursor position during paste** - keeps the current position of the cursor during a paste operation.
- **Use tab character** - The editor will write tabulator character after TAB key is pressed and these characters will not be replaced by spaces.
- **Show modules in separate tabs** - Displays tabs for each opened module (extension). When this option is not checked, it is possible to switch between module extensions by clicking the right mouse button on the tab of the file and choosing the appropriate extension or by clicking on the "Switch between module extensions" button in the toolbar menu.
- **Show line numbers** - Displays line numbers in the editor window.
- **Syntax highlighting** - Displays file content in specific colors with respect to file extension and compiler.
- **No horizontal scroll-bar** - Disables horizontal scroll-bar when line is not longer than window.
- **Outline current line** - Editor shows a frame around the current line.
- **Tab size** - number of spaces for tabulator.
- **Number of backup copies** - how many backup copies will be maintained for each saved document. The backup files are created within the same directory as the saved file. The name of file is the same, but there is '~' character added before the extension and a number of the backup copy is added at the end. The latest backup file has the number 0. The bigger the number is, the older is the backup.
- **Hint delay** - how long will an editor hint stay on the screen.


Editor Options

Edit Operations

- [ ] Preserve cursor position during paste
- [ ] Use tab character

Editor Options

- [ ] Show modules in separate tabs
- [ ] Show line numbers
- [x] Syntax highlighting
- [x] No horizontal scroll-bar
- [ ] Outline current line

Tab size: 8

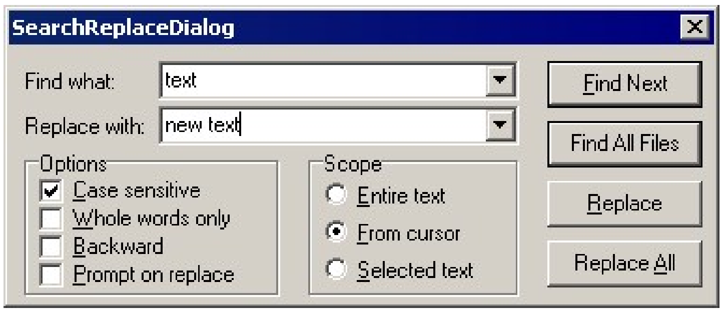
Number of backup copies: 0

Hint delay: 200 ms

Figure 2.51 - Editor Options

Search And Replace Dialog

This dialog window is invoked by the pop-up menu commands **Search | Find...** and **Search | Replace...**. It allows to specify the subject of search (or replace) and mode of operation.


SearchReplaceDialog

Find what: text

Replace with: new text

Options

- [x] Case sensitive
- [ ] Whole words only
- [ ] Backward
- [ ] Prompt on replace

Scope

- ○ Entire text
- ● From cursor
- ○ Selected text

Figure 2.52 - Search And Replace Dialog

Input Fields

- **Find what** - searched text
- **Replace with** - a new text that will replace the searched text in replace function.

Options

- **Case sensitive** - When checked, the case of letter of the searched text has to match.
- **Whole words only** - The searched text is found only as a whole word.
- **Backward** - direction of the search.

- **Prompt on replace** - specifies if the user will be asked about each item replacement during the replace process. (This option is available only if any replacement text is entered).

Scope

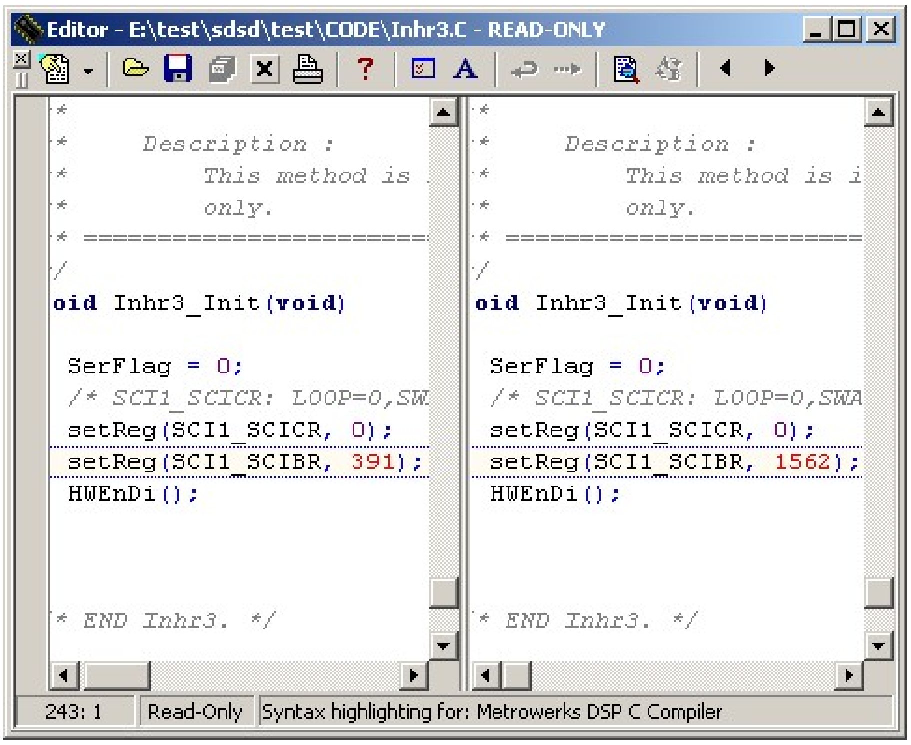
- **Entire text** - Entire text will be scanned.
- **From cursor** - Text from cursor to the end of file will be scanned only.
- **Selected text** - The current selection will be scanned only.

Buttons

- **Find Next** - Invokes the search process. The cursor will be placed on the next occurrence of the searched text. If the find/replace operations have not been done yet, the first occurrence of the text is found.
- **Find All Files** - Invokes the search process within all opened files. Places cursor in each file on the last occurrence of the searched text within the file.
- **Replace** - A next occurrence of the searched text is searched and if found, it is replaced by the replacement text.
- **Replace All** - All occurrences of the text within the current file are replaced by the searched text.

Comparison Mode

File editor in this mode has two panels showing a differences between the compared files. The **different lines are highlighted** with the light-yellow color and the different characters are red. The lines added to the file have a green background. To speed-up navigation between changes, the editor offer the **arrow buttons on the toolbar**. Pressing the right/left arrow button will move the cursor to the next/previous difference in the file.
This mode can be invoked automatically by Processor Expert in a case of comparing a bean modules with previously generated ones (See chapter *2.3.4 Beans Pop-up Menus* for details.) or by a 'DIFF' button when a changes tracking is enabled (See chapter *3.5.1.2 Tracking Changes in Generated Code* for details.)


Editor - E:\test\sdsd\test\CODE\Inhr3.C - READ-ONLY

```
*                                    *
*     Description :                  *     Description :
*         This method is             *         This method is i
*         only.                      *         only.
* ========================           * ========================
/                                    /
oid Inhr3_Init(void)                 oid Inhr3_Init(void)

 SerFlag = 0;                         SerFlag = 0;
 /* SCI1_SCICR: LOOP=0,SW             /* SCI1_SCICR: LOOP=0,SWA
 setReg(SCI1_SCICR, 0);               setReg(SCI1_SCICR, 0);
 setReg(SCI1_SCIBR, 391);             setReg(SCI1_SCIBR, 1562);
 HWEnDi();                            HWEnDi();



* END Inhr3. */                      * END Inhr3. */
```

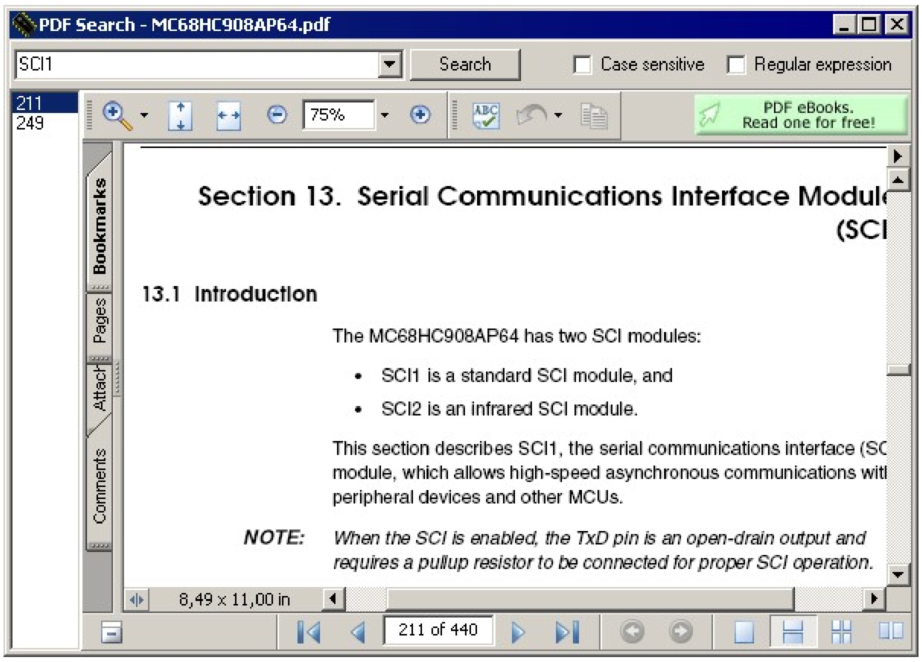
243: 1 | Read-Only | Syntax highlighting for: Metrowerks DSP C Compiler

Figure 2.53 - The comparison mode of editor

2.17. PDF Search

Processor Expert | Help | Search in PDF Documentation of the Target CPU

The **PDF Search** window allows the user to quickly browse PDF documentation for an CPU. It can also be invoked directly from the pop-up menu of the CPU bean.



Section 13. Serial Communications Interface Modul
(SCI

13.1 Introduction

The MC68HC908AP64 has two SCI modules:

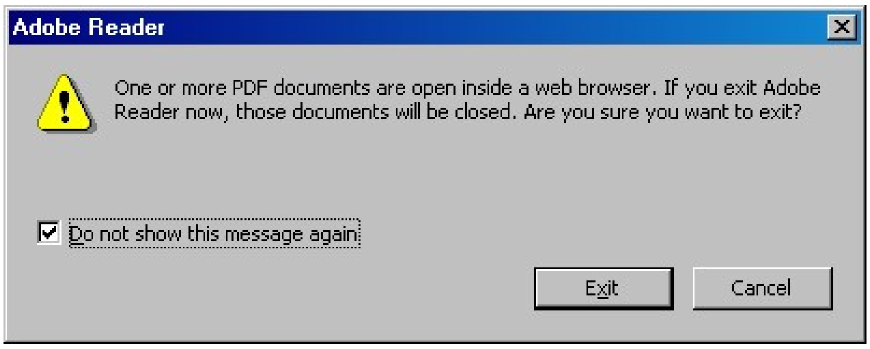
- SCI1 is a standard SCI module, and
- SCI2 is an infrared SCI module.

This section describes SCI1, the serial communications interface (SC module, which allows high-speed asynchronous communications wit peripheral devices and other MCUs.

NOTE: *When the SCI is enabled, the TxD pin is an open-drain output and requires a pullup resistor to be connected for proper SCI operation.*

Figure 2.54 - PDF Search Window

Notice for Acrobat Reader 6.0 users: *The Acrobat Reader can show the following dialog box while the PDF files are switched within the PDF search window:*



One or more PDF documents are open inside a web browser. If you exit Adobe Reader now, those documents will be closed. Are you sure you want to exit?

☑ Do not show this message again

*For a proper function of the PDF search the user should press the **Cancel** button.*

Three Panels Of The Window

- The **top panel** allows the user to specify the searched text/expression and enable/disable additional switches influencing the search process.
- On the **left side** of the widow is a narrow panel containing as a result of search a list of the pages containing the required information. Clicking on numbers in this listing invokes the appropriate page in the Adobe Acrobat Reader panel.
  Unfortunately Acrobat Reader does not permit highlighting of any text results found on the page. You may find on the page using the Acrobat Reader "Find" function (see Acrobat Reader toolbar).
- The **Acrobat Reader (or Adobe Reader) panel** shows PDF file content. This panel has it´s own toolbar which allows you to browse within the PDF file. This panel has also an internal Find function which is useful for locating searched phrase on the page. Using the toolbar you can print, move, and zoom in or out on the document. More documentation about controlling the Acrobat Reader plug-in can be found in Adobe Acrobat Reader documentation.

Additional Switches

- **Case sensitive** - If this switch is enabled, all letters of searched text must exactly match the text in the document including its case. If the switch is disabled, the case of the letters is ignored.
- **Regular expressions** - If this switch is enabled, the text entered as a searched phrase is treated as a regular expression. Users can use a power of regular expressions (For example it allows using a logical or set operators etc...). The syntax of regular expressions is a subset of commonly used Perl regular expressions. You can find more information on regular expressions and their syntax in chapter 2.17.1 Regular Expressions.

Our search technology uses the **PdfToText** program included in XPDF package. This package must be installed in a subdirectory named XPDF. The PDF documents are being converted to the text form in the background and stored in your TEMP folder. This conversion is executed only once for each PDF file. The conversion may take a moment: please wait until it is finished.

It is possible to find more information about XPDF on www.foolabs.com.
The configuration for the execution of the program PDFTOTEXT.EXE must be included in the Processor Expert Tools.

The Acrobat Reader must be installed on your computer as well.
The PDF Search feature was tested with Acrobat Reader 6.0 and 7.0

Opening CPU Documentation in Default PDF Viewer

To use a default PDF viewer instead of PDF search window for viewing the CPU documentation, set the option **Environment Options | Use default PDF viewer**

2.17.1. Regular Expressions

Regular Expressions are a widely-used method of specifying patterns of text to search for. Special metacharacters allow the user to specify, for instance, that a particular string the user is looking for occurs at the beginning or end of a line or contains *n* recurrences of a certain character.

Simple Matches

Any single character matches itself, unless it is a metacharacter with a special meaning as described below.

A series of characters matches that series of characters in the target string, so the pattern "bluh" would match "bluh" in the target string.

You can cause characters that normally function as metacharacters or escape sequences to be interpreted literally by 'escaping' them by preceding them with a backslash "\", for instance: metacharacter "^" matches the beginning of a string, but "\^" match character "^", "\\" match "\" and so on.

```
Examples:
  foobar          matches string 'foobar'
  \^FooBarPtr     matches '^FooBarPtr'
```

Escape Sequences

Characters may be specified using an escape sequence syntax much like that used in C and Perl: "\n" matches a newline, "\t" a tab, etc. More generally, \xnn, where nn is a string of hexadecimal digits, matches the character whose ASCII value is nn. If You need wide (Unicode) character code, You can use '\x{nnnn}', where 'nnnn' - one or more hexadecimal digits.

```
  \xnn     char with hex code nn
  \x{nnnn} char with hex code nnnn
  (one byte for plain text and two bytes for Unicode)
  \t       tab (HT/TAB), same as \x09
  \n       newline (NL), same as \x0a
  \r       car.return (CR), same as \x0d
  \f       form feed (FF), same as \x0c
  \a       alarm (bell) (BEL), same as \x07
  \e       escape (ESC), same as \x1b

  Examples:

  foo\x20bar   matches 'foo bar'
  (note space in the middle)
  \tfoobar     matches 'foobar' predefined by tab
```

Character Classes

You can specify a character class, by enclosing a list of characters in straight brackets [], which will match any one character from the list.

If the first character after the "[" is "^", the class matches any character not in the list.

```
Examples:
  foob[aeiou]r   finds strings 'foobar', 'foober' etc.
  but not 'foobbr', 'foobcr' etc.

  foob[^aeiou]r  find strings 'foobbr', 'foobcr' etc.
  but not 'foobar', 'foober' etc.
```

Within a list, the "-" character is used to specify a range, so that a-z represents all characters between "a" and "z", inclusive.

If you want "-" itself to be a member of a class, put it at the start or end of the list, or escape it with a backslash. If you want ']' you may place it at the start of list or escape it with a backslash.

```
Examples:
  [-az]      matches 'a', 'z' and '-'

  [az-]      matches 'a', 'z' and '-'
  [a\-z]     matches 'a', 'z' and '-'
  [a-z]      matches all twenty six small characters from 'a' to 'z'
  [\n-\x0D]  matches any of #10,#11,#12,#13.
  [\d-t]     matches any digit, '-' or 't'.
  []-a]      matches any char from ']'..'a'.
```

Metacharacters

Metacharacters are special characters which are the essence of Regular Expressions. There are different types of metacharacters, as described below.

Metacharacters - line separators

```
  ^      start of line
  $      end of line
  .      any character in line

Examples:

  ^foobar     matches string 'foobar' only if it's
     at the beginning of the line

  foobar$     matches string 'foobar' only if it's
```

```
     at the end of the line

  ^foobar$    matches string 'foobar' only if it's
     the only string in the line

  foob.r      matches strings like 'foobar', 'foobbr',
     'foob1r' and so on
Metacharacters - predefined classes

  \w     an alphanumeric character (including "_")
  \W     a nonalphanumeric
  \d     a numeric character
  \D     a non-numeric
  \s     any space (same as [ \t\n\r\f])
  \S     a non space

You may use \w, \d and \s within custom character classes.

Examples:

  foob\dr     matches strings like 'foob1r', ''foob6r'
  and so on but not 'foobar', 'foobbr' and so on

  foob[\w\s]r matches strings like 'foobar', 'foob r', 'foobbr'
  and so on but not 'foob1r', 'foob=r' and so on

Metacharacters - word boundaries

  \b     Match a word boundary
  \B     Match a non-(word boundary)
```

A word boundary (\b) is a spot between two characters that has a \w on one side of it and a \W on the other side of it (in either order), counting the imaginary characters off the beginning and end of the string as matching a \W.

Metacharacters - Iterators

Any item of a regular expression may be followed by another type of metacharacters - iterators. Using this metacharacters you can specify a number of occurrences of a previous character, metacharacter or subexpression.

```
  *      zero or more, similar to {0,}
  +      one or more, similar to {1,}
  ?      zero or one, similar to {0,1}

  {n}    exactly n times
  {n,}   at least n times
  {n,m}  at least n but not more than m times
```

So, digits in curly brackets of the form {n,m}, specify the minimum number of times to match the item n and the maximum m. The form {n} is equivalent to {n,n} and matches exactly n times. The form {n,} matches n or more times. There is no limit to the size of n or m, but large numbers will chew up more memory and slow down regular expressions execution.

If a curly bracket occurs in any other context, it is treated as a regular character.

```
Examples:
  foob.*r     matches strings like 'foobar',
     'foobalkjdflkj9r' and 'foobr'

  foob.+r     matches strings like 'foobar',
     'foobalkjdflkj9r' but not 'foobr'

  foob.?r     matches strings like 'foobar', 'foobbr'
     and 'foobr' but not 'foobalkj9r'

  fooba{2}r   matches the string 'foobaar'

  fooba{2,}r  matches strings like 'foobaar', 'foobaaar',
     'foobaaaar' etc.

  fooba{2,3}r matches strings like 'foobaar', or 'foobaaar'
     but not 'foobaaaar'
```

Metacharacters - Alternatives

You can specify a series of alternatives for a pattern using "|" to separate them , so that fee|fie|foe will match any of "fee", "fie", or "foe" in the target string (as would f(e|i|o)e). The first alternative includes everything from the last pattern delimiter ("(", "[", or the beginning of the pattern) up to the first "|", and the last alternative contains everything from the last "|" to the next pattern delimiter. For this reason, it's a common practice to include alternatives in parentheses to minimize confusion about where they start and end.

Alternatives are tried from left to right, so the first alternative found for which the entire expression matches, is the one that is chosen. This means that alternatives are not necessarily greedy. For example: when matching foo|foot against "barefoot", only the "foo" part will match, as that is the first alternative tried, and it successfully matches the target string. (This might not seem important, but it is important when you are capturing matched text using parentheses.)

Also remember that "|" is interpreted as a literal within square brackets, so if you write [fee|fie|foe] You're really only matching [feio|].

```
Examples:
  foo(bar|foo)  matches strings 'foobar' or 'foofoo'.
```

Metacharacters - Subexpressions

The bracketing construct (...) may also be used for defining regular expression subexpressions. Subexpressions are numbered based on the left to right order of their opening parenthesis. The first subexpression has the number '1'.

```
Examples:
(foobar){8,10}  matches strings which contain
   8, 9 or 10 instances of the 'foobar'

foob([0-9]|a+)r matches 'foob0r', 'foob1r' ,
'foobar', 'foobaar', 'foobaar' etc.
```

Metacharacters - Backreferences

```
Metacharacters \1 through \9 are interpreted as backreferences.
\<n> matches previously matched subexpression #<n>.

Examples:

  (.)\1+         matches 'aaaa' and 'cc'.
  (.+)\1+        also match 'abab' and '123123'
  (['"]?)(\d+)\1 matches '"13" (in double quotes), or '4'
 (in single quotes) or 77 (without quotes)etc
```

3. Application Design

This chapter should help users to design application using Processor Expert and Embedded Beans. You will find here recommendations and solutions helping you to write and optimize a code effectively and properly. If you are a beginner, please see the section Quick start that shows how to generate the code of your first project.

The following subchapters explain

- Quick Start in Processor Expert
- Basic Principles
- Configuring Beans
- Implementation Details
- Code Generation
- Embedded Bean Optimizations
- Low-level Access to Peripherals

3.1. Quick Start in Processor Expert

Step 1 - Open an example

You may start learning Processor Expert by opening one of the available examples. All Processor Expert examples are accessible from the CodeWarrior.

- ***In CodeWarriror for HC(S)08:***
  On the startup dialog click on the button **Load Example Project**. If the CodeWarrior is already opened, invoke the startup dialog by using the menu command **File | Startup Dialog**. Then unfold the CPU family and category folder and select an example (Processor Expert examples are listed in the folder *Processor Expert Examples*). Then specify a name of the new project and its location. Processor Expert will start with a new project based on the example project.

- ***In CodeWarriror for HC(S)12:***
  To open an example select the command **CodeWarrior main menu | File | Open...** and find the directory ***{CodeWarrior}*****\ProcessorExpert\projects\HCS12\Demo.Tutorial** (where the {CodeWarrior} is the path where the CodeWarrior is installed into) and select the file **LED.mcp**.

- ***In CodeWarrior for 56F800/E:***
  To open an example select the command **CodeWarrior main menu | File | New...** or use the **New** icon on the toolbar. Select the **Processor Expert Examples stationery** and select a desired example. Then specify a name of the new project. Processor Expert will start with a new project based on the example project.

Step 2 - Code generation

After opening an example, you need invoke the code generation of the project to obtain all sources. Select command **CodeWarrior Main Menu | Processor Expert | Generate Code 'Project Name'**. After the code generation, the bean source modules will be inserted into the ***Generated Code*** folder in the CodeWarrior project window and the event and main source modules will be inserted into the ***User Modules*** folder in the CodeWarrior project window. Generated source code can be displayed in editor by the left mouse button double-click on selected module in the project window.

Step 3 - More Complicated Example

Once you have learned the basic skills, you may open a more complicated example in order to get to more advanced level of code generation.

Creating New Projects

See the chapter 4 Processor Expert Tutorials for step-by-step tutorials on creating Processor Expert projects from the beginning.

3.2. Basic Principles

The application created in Processor Expert is built from the building blocks called Embedded Beans. The following sub-chapters describe the features of Embedded Beans (and the CPU beans - special type of Embedded Beans) and what they offer to the user.

- Embedded Beans
- CPU Beans

3.2.1. Embedded Beans

Embedded beans encapsulate the initialization and functionality of embedded systems basic elements, such as CPU core, CPU on-chip peripherals (for details on categories of beans delivered with Processor Expert see chapter 3.2.1.1 Bean Categories), FPGAs, standalone peripherals, virtual devices and pure software algorithms.
These facilities are interfaced to the user via properties, methods and events. It is very similar to objects in Object Oriented Programming (OOP) concept.

Easy Initialization

A user can initialize beans by setting their initialization properties in the Bean Inspector. Processor Expert generates the initialization code for the peripherals according to the properties of the appropriate beans. User can decide whether the bean will be initialized automatically at startup or manually by calling the bean's Init method.

Easy On-chip Peripherals Management

Processor Expert knows exactly the relation between allocated peripherals and selected beans. When the user chooses a peripheral in the bean properties, Processor Expert proposes all the possible candidates but signals which peripherals are allocated already (with the icon of the bean allocating the peripheral) and also signalizes peripherals that are not compatible with current bean settings (with a red exclamation mark). In the case of an unrealizable allocation, an error is generated.

Unlike common libraries, Embedded Beans are implemented for all possible peripherals, with optimal code. The most important advantages of the generated modules for driving peripherals are that you can:

- Select any peripheral which supports bean function and change it whenever you want during design time.
- Be sure that the bean setting conforms to peripheral parameters.
- Choose the initialization state of the bean.
- Choose which methods you want to use in your code and which event you want to handle.
- Use several beans of the same type with optimal code for each bean.

The concept of the peripheral allocation generally does not enable sharing of peripherals because it would make the application design too complicated. The only way to share resources is through the beans and their methods and events. For example, it is possible to use the RTIshared bean for sharing periodic interrupt from timers.

Methods

Methods are interfacing bean functionality to user's code. All enabled methods are generated into appropriate bean modules during code generation process. All Methods of each bean inserted into the project are visible as a subtree of the beans in the Project panel.
You can use in your code all enabled methods. The easiest way to call any method from your code is to drag and drop the method from project panel into the editor. The complexity and number of methods depend on the bean's level of abstraction.

Events

Some beans allow handling the hardware or software events related to the bean. The user can specify the name on function invoked in the case of event occurrence. They are usually invoked from the internal interrupt service routines generated by Processor Expert. If the enabled event handling routine is not already present in the event module then the header and implementation files are updated and an "empty" function (without any code) is inserted. The user can write event handling code into this procedure and this code will not be changed during the next code generation.
All Methods and Events of each bean inserted into the project are visible as a subtree of beans in the Project panel.

Interrupt Subroutines

Some beans, especially the Low-level beans and Peripheral Initialization beans (please see more details in chapter 3.2.1.1 Bean Categories) allow to assign an interrupt service routine name to a specific interrupt vector setup.
The name of the Interrupt service is generated directly to the interrupt vector table and the user has to do all necessary control registers handling within his/her code. See chapter *3.5.3.1 Typical Usage of Peripheral Initialization Beans* for details.
ISRs items are listed in subtree of a bean in the Project panel.

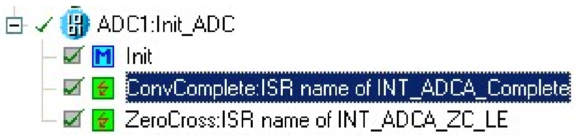

ADC1:Init_ADC
Init
ConvComplete:ISR name of INT_ADCA_Complete
ZeroCross:ISR name of INT_ADCA_ZC_LE

Figure 3.1 - Example Of a Bean With Two ISRs

Highly Configurable and Extensible Library

Embedded Beans can be created and edited manually or with the help of Bean Wizard. CPU Beans are a special category of beans.

More information about Embedded beans can be found in the Processor Expert:

Help | Processor Expert | Embedded Beans
Help | Processor Expert | Supported CPUs

3.2.1.1. Bean Categories

Bean Selector Categories

Complete list of bean categories and corresponding beans can be found in the Bean Selector (main menu | View | Bean Selector) at the page "Bean Categories".

The categories are related to the bean's functionality and allow to quickly find appropriate bean for a desired function. There are the following four main categories containing many sub-categories.

- **CPU** - all available CPU beans. The CPU folder in Bean Selector contains subfolders for the CPU families.
- **CPU External Devices** - beans for devices externally controlled to the CPU. For example sensors, memories, displays or EVM equipment.
- **CPU Internal Peripherals** - beans using any of on-chip peripherals offered by the CPU. The Bean Selector folder with the same name contains sub-folders for the specific groups of functionality. (i.e. Converters, Timers, PortIO etc.)

  Note: It seems that beans (especially in this category) correspond to on-chip peripherals. Even this declaration is close to the true, the main purpose of the bean is providing the same interface and functionality for all supported CPU derivatives. This portability is the reason why the bean interface often doesn't copy all features of the specific peripheral.

- **SW** - beans encapsulating a pure software algorithms or inheriting a hardware-dependent beans for accessing peripherals. These beans (along with beans created by the user) can be found in a bean selector folder SW.

Specific functionality of the CPU derivative may be supported as a **version-specific** settings of the bean. For more about this feature please refer to Version specific in the bean documentation or Beans Implementation help chapter.

Levels of Abstraction

Processor Expert provides beans with a different level of abstraction and settings comfort.

- **High Level Beans** - The beans that are the basic set of beans designed carefully to provide functionality of most microcontrollers on the market. An application built from these beans can be easily ported to another microcontroller supported by the Processor Expert. This basic set contains beans for simple I/O operations (BitIO, BitsIO, ByteIO, ...), timers (EventCounter, TimerInt, FreeCntr, TimerOut, PWM, PPG, Capture, WatchDog,...), communication (AsynchroSerial, SynchroMaster, SynchroSlave, AsynchroMaster, AsynchroSlave, IIC, ADC, internal memories, etc...).
  This group of beans allows comfortable settings of a desired functionality such as time in ms or frequency in Hz, the user doesn't have to know the details about hardware registers. CPU specific features are supported only as CPU specific settings or methods and are not portable. See chapter *2.5.3.3 Version Specific Items* for details.
  The beans inheriting or sharing a high level bean(s) to access hardware are also high level beans.
- **Low Level Beans** - The beans that are dependent on the peripheral structure to allow the user to benefit from non-standard features of a peripheral. The level of portability is decreased due to a different bean interface and the bean is usually implemented only for a CPU family offering the appropriate peripheral. However, there is still implemented a comfortable settings of devices' features and effective set of methods and events.
- **Peripheral Initialization Beans** - These beans are on the lowest level of abstraction. An interface of such beans is based on the set of peripheral control registers. These beans cover all features of the peripherals and were designed for initialization of these peripherals. Usually contain only "Init" method, see 3.5.3.1 Typical

Usage of Peripheral Initialization Beans for further details). The rest of the function has to be implemented using a low level access to the peripheral. This kind of beans could be found in the "**CPU Internal Peripherals / Peripheral Initialization Beans**" folder of the Bean selector and they are available only for some CPU families. The interface of these beans might be different for a different CPU. The name of these beans starts with the prefix 'Init_'.

Table of the features of the different levels beans

Feature	**High level**	**Low level**	**Peripheral Init**
High level settings portable between different CPU families	yes	partially	no
Portable method interface for all CPU families	yes	partially (usually direct access to control registers)	*Init* method only
CPU specific peripheral features support	partially	mostly yes	full
Low level peripheral initialization settings	no	partially	yes
Speed mode independent timing	yes	mostly yes	no
Events support	yes	yes	no (direct interrupt handling)

3.2.2. CPU Beans

Processor Expert | Help | Supported CPUs, Compilers and Debuggers

A CPU bean is an Embedded Bean encapsulating one CPU type. As with all other beans, CPU Beans have properties, methods and events. A Processor Expert project may contain several CPU beans. The project generated for one CPU is called an Application. CPUs included in a project are displayed in the upper part of the Project panel (depending on the Project panel settings); just one of them must be active (selected as the **Target CPU**) before starting the code generation.

The **Build options** accessible in the Bean Inspector of the CPU bean allow to set properties of the **Compiler** and **Debugger** (if it is supported).

In the CPU Bean Inspector (page **Used**) it is also possible to select the peripherals which cannot be used by Processor Expert. These peripherals will not be available for the beans in the project and may be freely used by any external module.

Portability

- It is possible to change the target CPU during the development of an application and even to switch between multiple CPUs. This can be done simply by adding another CPU to the project and selecting it as the target CPU.
- To connect the new CPU peripherals to the application beans correctly, it is possible to specify CPU on-chip peripheral names (See chapter *3.2.2.3 Changing Names of Peripheral Devices* for details.). *This way the same peripheral could be used on different CPU derivatives even if the original name is different.*
- Specific application options for single targets are set in Project Options

Note: CPU peripherals names and Application Options are in the same popup menu as CPU inspector. See the last part of this page to learn how to open it.

How to Add a CPU to the Project.

- In the Bean Selector window, select the **CPU** category and find the desired CPU bean.
- Double-click the desired CPU icon to add it into the project. When the CPU bean is added, it appears in the upper part of the Project panel. If selected as the target CPU, the processor will be displayed in the Target CPU window.

How to Select a CPU as the Target CPU.

The first CPU added to the project is automatically selected as the **target CPU**. It means that code will be generated for this CPU. When there is more than one CPU in the project, the target CPU can be changed this way:

- Move the mouse pointer to the CPU icon on the Project panel..
- Click with the right mouse button - a popup menu will appear.
- Click "Select CPU as target" - the CPU is selected as target.

This setting doesn't affect the setting of the target in the Targets tab in CodeWarrior's project panel. When the user changes the target CPU in Processor Expert project panel and the CPU doesn't match with the current CodeWarrior target settings the linker dialog is invoked during the code generation allowing the user to switch the linker settings.

How to Change CPU Settings.

The only way to modify CPU settings (properties, methods, events, chip selects, timing, user-reserved peripherals, also compiler and debugger settings, etc.) is to invoke the Bean Inspector for the selected CPU.

You can do that with the following steps (assuming that the CPU is included in the project):

- Command in CPU pop-up menu:
  a. Move the mouse pointer to the CPU icon in the Project panel.
  b. Click with the right mouse button - a popup menu will appear
  c. Click **CPU inspector** - the CPU Bean Inspector will appear
- Or use double-click on the CPU icon in the Project panel.

Look at the given CPU help page to learn about what each feature and setting means.
For a detailed description of the current CPU properties, methods and events, select **Bean Inspector | Help | Help on Bean** in Bean Inspector.

3.2.2.1. CPU Properties Overview

CPU Properties can be set in CPU Bean Inspector. The complete list of CPU properties and its description is available in the help page for the CPU (**Bean Inspector | Help | Help on Bean**).

Settings of these properties define the basic settings of the CPU:

- CPU type
- external Xtal frequency (and sub-clock xtal frequency)
- PLL settings
- initialization interrupt priority
- external bus and signals
- speed modes (See the following chapter Speed Modes).
- and all other functions which are not directly encapsulated by beans

3.2.2.2. Speed Modes Support

The CPU bean supports up to three different **speed modes**. The three **speed modes** are a Processor Expert -specific concept which (among all the other PE features and concepts) ensures the portability of the PE projects between different CPU models.

In fact, the 3 speed modes are a generalization of all the possible CPU clock speed modes used for power-saving that can be found in most of modern microcontrollers. In the area of embedded systems, power saving and power management functions are so important that we could not neglect the proper HW- independent software implementation of these functions.
Therefore, for keeping the portability (HW independence) of PE projects, we recommend not to program the CPU speed functions manually, but use these 3 CPU Bean speed modes instead:

- **High speed mode** - this mode is selected after reset and must be enabled in the project. This speed mode must be the fastest mode of the main CPU clock.
- **Low speed mode** - this mode is usually used for another PLL or main prescaler settings of the main CPU clock.
- **Slow speed mode** - this mode is usually used for the slowest possible mode of the main CPU clock.

The modes can be switched in the runtime by the following CPU Bean methods: (*SetHighSpeed, SetLowSpeed and SetSlowSpeed*). If a speed mode is enabled in the CPU Bean properties, the corresponding method will be enabled automatically.

For each bean in the project, whose function is dependent on the CPU timing:

- Using the bean property "CPU clock/speed selection", it is possible to define the speed modes that the bean supports. If the CPU speed mode is changed to a mode which the bean does not support for some reason, the bean will be disabled right after the CPU speed mode is changed. Otherwise the bean will be enabled.
- During the design, all the timing-related settings for such a bean are checked to be correct in all the speed modes that the bean supports (is enabled in).

- If the speed mode is changed, the current settings for the bean will be conserved.
- Before or after the speed mode is changed, the following event functions are called: *BeforeNewSpeed, AfterNewSpeed.*

3.2.2.3. Changing Names of Peripheral Devices

The editor window lists all **peripherals names**. Initially, on-chip peripheral devices (pins, ports, timers, converters, etc.) have default catalogue names. You may change these names using this editor.

How to Open the Editor.

In order to open the CPU Peripherals Names Editor, follow these steps:

- Move the mouse pointer to the selected CPU icon on the Project panel.
- Press the right mouse button - a popup menu will appear.
- Click on "**CPU peripherals names**".

Purpose of name changes.

Some beans are associated with on-chip The link between a bean and a peripheral is done by names (not by any internal identification). Therefore, if you change the name of a peripheral to which a bean was linked, an error will occur. The bean's link will no longer be valid.

Fortunately, when you change a peripheral's name, Processor Expert offers to update the name in all the project's beans.

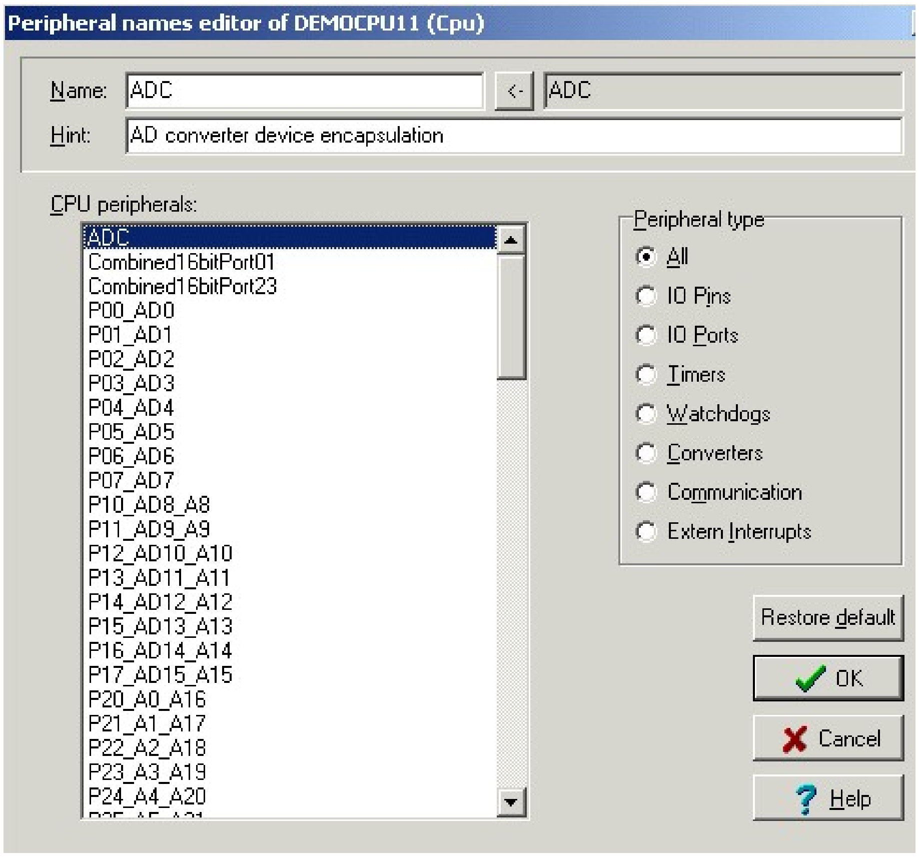
Advantages of the option to change names:

- Peripheral names may express the reality of the CPU environment, making the project more understandable.
- When working with several processors (having different peripheral names), one may assign a project-specific name in order to facilitate the CPU changes in the project.
  Example: **Timer0** of CPU1 and **TmrA** of CPU2 have the same function in the project. They are both associated with **Bean1:TimerOut**. Every time you change the Target CPU from CPU1 to CPU2 and vice versa, you need to modify in the Bean Inspector the name of the peripheral linked to the **TimerOut** bean. To avoid this, you can rename Timer0 and TmrA to one common name - **MyTimer** - and link the bean (Bean1) to the **MyTimer** peripheral. Now, when CPU1 will be set active (selected as target CPU), Bean1 will be linked to Timer0, and when CPU2 will be set active, Bean1 will be linked to TmrA.

How to Edit CPU Peripheral Names.

Follow these steps:

- Select a peripheral type in order to display all the CPU peripherals of this type.
- In the Name edit box, insert a new name for the selected CPU peripheral.
  Note: If you want to get back the default catalogue name, press the left-arrow button.
- Press the OK button.
- Answer the confirmation request of the dialog box.



Name: ADC ADC

Hint: AD converter device encapsulation

CPU peripherals:

```
ADC
Combined16bitPort01
Combined16bitPort23
P00_AD0
P01_AD1
P02_AD2
P03_AD3
P04_AD4
P05_AD5
P06_AD6
P07_AD7
P10_AD8_A8
P11_AD9_A9
P12_AD10_A10
P13_AD11_A11
P14_AD12_A12
P15_AD13_A13
P16_AD14_A14
P17_AD15_A15
P20_A0_A16
P21_A1_A17
P22_A2_A18
P23_A3_A19
P24_A4_A20
```

Peripheral type: [✓ All] IO Pins IO Ports Timers Watchdogs Converters Communication Extern Interrupts

Figure 3.2 - CPU Peripheral Names Editor

3.3. Configuring Beans

Configuring the beans in the project is one of the main activities in Processor Expert. It influences the initialization, run-time behavior and range of functionality available to the user of the generated code. For the description of the user interface of the beans settings please see the chapters 2.3 Project Panel and 2.5.3 Bean Inspector.

This following sub-chapters describe hints and information that should allow the user to correctly and effectively configure the Embedded Beans used in the project.

- Interrupts and Events
- Configurations
- Design Time Checking: Consequences and Benefits
- Timing Settings
- Creating User Bean Templates
- Signal Names
- Bean Inheritance and Bean Sharing
- Pin Sharing
- Import 56800/E Project From Quick-Start

3.3.1. Interrupts and Events

This chapter describes the details of interrupt and events processing in the code generated by Processor Expert.

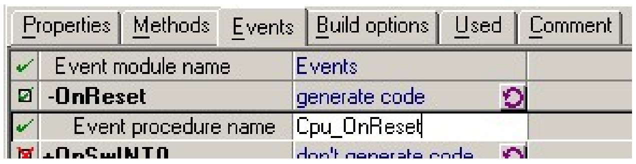
An **Interrupt** is a signal that causes the CPU stop the execution of a code is suspended(the state of the CPU core is saved on the stack) and Interrupt service routine is executed instead. After the execution is finished the suspended program continues on the place where it was interrupted (the previous state of the CPU core is restored from the stack). The signals causing interrupts can be a hardware events or software commands. For more details please see an appropriate CPU manual.
Each interrupt can have assigned an **Interrupt Service Routine** (ISR) that is called when the interrupt occurs. The table assigning the subroutines to interrupts is called **Interrupt Vector Table** and it is completely generated by Processor Expert. See chapter *3.3.1.1 Interrupt Vector Table* for details. Most of the interrupts have a corresponding Processor Expert Events that allows to handle these interrupts.

Processor Expert **Events** are part of the Embedded bean interface and encapsulate the hardware or software events within the system. Events are offered by the High and Low Level beans to help the user to service the events without a knowledge of the platform specific code needed for such service.

Processor Expert **Events** can be enabled and disabled and have a user written program subroutines that are invoked when the event occurs. Events often correspond to interrupts and are for that case invoked from the generated ISR. Moreover, the event can also be a pure software event caused by a buffer overflow or improper method parameter.

Enabling Event

Functionality of each event can be enabled or disabled. The user can easily enable the event and define its name within the Bean Inspector of the appropriate bean. Another possibility is to double-click an event icon in the bean's subtree or use a pop-up menu in the Project Panel window.



✓	Event module name	Events
☑	**-OnReset**	generate code
✓	Event procedure name	Cpu_OnReset
☒	**+OnSwINT0**	don't generate code

Figure 3.3 - Event Example in the Bean Inspector Events Tab

Writing an Event Handler

Event handler is a subroutine that is assigned to a specific event. After the event is enabled Processor Expert generates the empty function of the specified name to the Event module. See chapter *3.5.1 Code Generation* for details. The user can directly open the Event code (when it already exists) using a **Bean pop-up menu | View/Edit event module** or a double-click on the event within a Project Panel. It is an ordinary function and the user needs not to provide the interrupt handling specific code in his/her event code.

Interrupt Service Routines

When **High or Low level beans** are used, the interrupts functionality is covered by the events of the beans. The interrupt subroutines calling the user's event handlers are generated to the bean modules and PE provides parts of the background code necessary to correctly handle the interrupt requests.

The **Peripheral Initialization beans** can only provide the initialization of the interrupt and generate a record to the Interrupt Vector Table. The user has to provide a full implementation of the interrupt subroutine. See chapter *3.5.3.1 Typical Usage of Peripheral Initialization Beans* for details.

3.3.1.1. Interrupt Vector Table

An **interrupt vector** is a pointer to an interrupt handling subroutine.
An **Interrupt Vector Table** (IVT) is a table (list) which contains single interrupt vectors. Each interrupt corresponds to one interrupt vector. IVT may be:

- placed in **ROM** - usually first-level IVTs: the jump to the right vector is done by hardware, no interrupt vectors can be changed at runtime.
- placed in **RAM** - usually second-level IVTs: the jump to the right vector is done by software first-level interrupt handling subroutine. This implementation can be slower due to this software redirection to users routine.
  The interrupt vectors placed in RAM and **not allocated by any bean** can be changed at runtime using CPU bean methods (GetIntVect and SetIntVect).

  Notice: Interrupt vectors in RAM are available only in the HC(S)08 and HC(S)12 & HCS12X versions.

The type of the IVT (ROM/RAM) can be setup using the option **Application Options | Interrupt Vector Table** .

Each processor that can handle and process interrupts has a first-level IVT (in ROM). The second-level is used only when it is necessary (i.e. IVT is placed in RAM).

Processor Expert generates content of the whole IVT into the file **vectors.c**. The content of the file depends on the compiler syntax of the interrupt declarations.

3.3.1.2. Processor Expert Priority System

Some CPUs support selectable **interrupts priorities**. The user may select a priority for each interrupt vector. The interrupt with a higher priority number can interrupt a service routine with the lower one.

Processor Expert supports the following settings in design-time: interrupt priority and priority of the event code. Priority can be changed also in the user code. The user may use Cpu method to adjust the requested value.

Small microcontroller architectures support only a basic interrupt control: interrupts *enabled* or *disabled*. Settings of the interrupt priority may be ignored for such microcontrollers. The only option is to enable interrupts for the user event code.

Interrupt Priority

The user may select interrupt priority in the bean properties, just below the interrupt vector name. Processor Expert offers the following values, which are supported for all microcontrollers:

- minimum priority
- low priority
- medium priority
- high priority
- maximum priority

The selected value is automatically mapped to the priority supported by the target microcontroller. It is indicated in the third column of the Bean Inspector.
The user may select a target-specific value (such as priority 255), if portability of the application to another architecture is not required.

Priority of Event Code

The user can also select a priority for the processing of his/her event code. This setting is available for the events that are invoked from the Interrupt Service Routines. This priority may be different from the interrupt priority. However, the meaning of the number is the same - the event may be interrupted only by the interrupts with the higher priority. Processor Expert offers the following architecture independent values:

- same as interrupt - default value which means that Processor Expert doesn't generate any code influencing the priority of the event - the priority is in the state determined by the default hardware behavior.
- minimum priority
- low priority
- medium priority
- high priority
- maximum priority
- interrupts disabled - *e.g. the highest priority supported by the microcontroller, which may be interrupted only by non-maskable interrupts.*

The selected value is automatically mapped to the priority supported by the target microcontroller and the selected value is displayed in the third column of the Bean Inspector.
Please see version specific information below. The user may also select a target-specific value, if portability of the application to another architecture is not required.

Note: Some events do not support priorities, because their invocation is not caused by the interrupt processing.

Version specific information for HCS12X derivatives
Processor Expert offers the following event priority options:

- **Interrupts Enabled** - interrupts are enabled and the interrupts with the higher priority than the current interrupt priority can interrupt the event code. (The state of the register CCRH is not changed.)
- **Interrupts Disabled** - all maskable interrupts are disabled. (The state of the register CCRH is not changed.)
- **0** - The same as Interrupts Disabled
- **1..7** - Priorities from lowest (1) to highest (7). The code generated by Processor Expert before the event invocation sets the event code priority to the specified value (by writing to the CCRH register) and enables

interrupts.

- **Same as interrupt** - default behavior of the architecture - no interrupts can interrupt the event. The same as Interrupts Disabled.
- Other values are mapped to the priorities 1..7.

Version specific information for HCS12 derivatives
Processor Expert offers the following event priority options:

- **Interrupts Disabled** - all maskable interrupts are disabled.
- **Interrupts Enabled** - all maskable interrupts are enabled.
- **Same as interrupt** - default behavior of the architecture - no interrupts can interrupt the event. The same as Interrupts Disabled.

Version specific information for 56800 derivatives
Processor Expert offers the following event priority options:

- **Interrupts Enabled** - interrupts are enabled so the event routine can be interrupted by another interrupt.
- **Interrupts Disabled** - all maskable interrupts are disabled.
- **Same as interrupt** - default behavior of the architecture within interrupts service routines - interrupts are disabled.

Version specific information for 56800E derivatives
Processor Expert offers the following event priority options:

- **Interrupts Disabled** - all maskable interrupts are disabled within the event routine.
- **1..3** - Priorities from the lowest (1) to the highest (3). The code generated by Processor Expert before the event invocation sets the event code priority to the specified value. The event routine can be interrupted only by a higher priority interrupt than the specified number.
- **Same as interrupt** - The priority of the event routine stays on the level set for the interrupt. The event routine can be interrupted only by a higher priority interrupt then the value set for the interrupt.
- Other values are mapped to the priorities 1..3.

3.3.2. Configurations

The user can have several configurations of the project in one project file. The configuration system is very simple. Every configuration keeps the **enable/disable state of all beans** in the project (it does NOT keep bean settings!). If you enable/disable a bean in the project, the bean state is updated in the currently selected configuration. If you create a new configuration the current project state is memorized.

Configurations of the current project are listed in the Project Panel configurations folder. They can be managed using a pop-up menu of the specific configuration or the whole Configurations folder.

Configuration properties and a user comment can be changed in the configuration inspector. See chapter *2.5.4 Configuration Inspector* for details.

The symbol for conditional compilation is defined if it is supported by selected language/compiler. The symbol **PEcfg_*[ConfigurationName]*** is defined in the CPU interface.
The user can switch using this symbol between variants of code according to the active configuration (see example in this chapter).

Configuration also stores which CPU is selected as the target CPU.
If the name of the configuration matches the name of one of the CodeWarrior's targets, this target is automatically selected as an active target when the user runs code generation.

Note: It is possible to have two beans with the same name in Project. Each of the beans could be enabled in different configuration. This way the user can have different setup of a bean (a bean with the same name) in multiple configurations.

Example

Let's have a configuration named 'Testing case'. We would like to use a bean and part of our code using the bean only in the 'Testing case' configuration. We make the Testing case configuration active. After the successful code generation the CPU.H file contains the following definition:

```
/* Active configuration define symbol */
#define PEcfg_Testingcase 1
```

We will surround the part of the code used only in this configuration the following way:

```
...
#ifdef PEcfg_TestingCase
   Bean_MethodCall(...);
#endif
...
```

3.3.3. Design Time Checking: Consequences and Benefits

During the design time, Processor Expert performs instant checking of the project. As a result of this checking, error messages may appear in the Error Window or directly in the third column of the Bean Inspector (on the faulty items line). Sometimes, it may happen that only one small change in the project causes several (general) error messages. The most common reasons for this behavior are stated below.

On-Chip Peripherals

Some beans use on-chip peripherals. In the Bean Inspector you can choose from all possible peripherals that can be used for implementation of the function of the current bean. Processor Expert provides checking for required peripheral features such as word width and stop bit for serial channel, pull resistor for I/O pin and others.
Processor Expert also protects against the use of one peripheral in two beans. If the peripheral is allocated for one bean then the settings of this peripheral cannot be changed by any other bean. **The state of an allocated peripheral should never be changed directly in the user code. (Using special registers, I/O ports etc.) We recommend to always use methods generated by Processor Expert.** If the functionality of generated methods is not sufficient for your application, you can use PESL (Processor Expert System Library). See chapter *3.7 Low-level Access to Peripherals* for details.

Note that if a peripheral is allocated to any bean, all its parts are reserved. For example if you use the 8-bit I/O port, all the I/O pins of the port are allocated and it is not possible to use them in other beans.
In some timer beans you can choose if you want to use only a part of the timer (compare register) or an entire timer. If you select the entire timer, the driver can be optimized to best work with the timer: it can, for example, invoke reset of the timer whenever is it needed by the bean function.

Interrupt Priority

If the target CPU shares interrupt priority between several interrupt vectors or shares interrupt vectors, Processor Expert provides checking of interrupt priority settings. If you would like to have more detailed information about Interrupt Priority see the Priorities page.

Memory

Processor Expert always checks the usage of internal and external memories accessible via CPU address and data bus. Position and size of internal memory is defined by the CPU type and can be configured in the CPU Properties (if supported). External memories must be defined in CPU Properties.
Any bean can allocate a specified type of memory. See bean descriptions for detailed information about requirements for types of memory. Processor Expert provides checking of memory and protects you from making a poor choice. (*For example: if a bean requires external Flash, it is not possible to enter an address in internal RAM*).

The bits can also allocate memory. Therefore you can be sure that only one bean uses an allocated bit of a register in external address space.

Timing

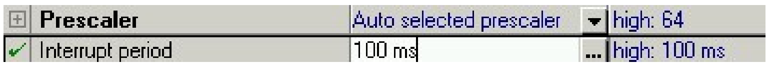
The settings of all timed high-level beans are checked using the internal timing model. See chapter *3.3.4 Timing Settings* for details. If there is no error reported, it means that Processor Expert was successful in calculating the initialization and runtime control values for all beans so the settings shall work according to the configuration.

3.3.4. Timing Settings

Many high-level beans contain a timing configuration (e.g. speed of the serial communication, period of the interrupt, conversion time of the ADC etc). Processor Expert allows to configure such timing using user-friendly units and it also contains a model of the complete MCU timing which allows to calculate the required values of control registers and it also allows a continuous validation of the timing settings.

Timing Model

A bean timing can be viewed like a chain of elements (dividers, multipliers etc.) between the main clock source and the device configured by the bean. The user sets the desired timing value using the Timing Dialog. See chapter *2.5.3.1 Dialog Box for Timing Settings* for details. Processor Expert tries to configure individual elements in the timing chain to achieve the result and the user is informed if it was successful. After leaving the Timing Dialog, the real value of the prescaler and timing are shown in the third column of the bean inspector.



Prescaler	Auto selected prescaler	high: 64
Interrupt period	100 ms	high: 100 ms

Figure 3.4 - Example of the timing and prescaler setup

The complete MCU timing model with the current state of the individual elements can be viewed using the CPU Timing Model Window. See chapter *2.8 CPU Timing Model* for details.

Timing Setup Problems

The errors are reported red in the Timing Dialog or in the timing property line in the Bean Inspector. The error summary is available in the Error window (see details in chapter 2.6 Error Window). Please follow the error message text to find the source of the problem. If no error is reported, it means that Processor Expert can achieve the desired timing for all beans in the project.

Common problems that make impossible to set a timing value:

- *It is impossible to set some item(s).*
  This problem is reported in the bean or the timing dialog and the user is informed which value has incorrect value. The reason is usually the hardware structure and limitations of the CPU. The timing dialog shows the a list of values (ranges) that are allowed to be set. There also might be necessary to increase the allowed error (using the 'Error' field within the Timing Dialog) that specifies the allowed difference between the required value and possible value that can be produced by the hardware.
- *Settings for the device are mutually incompatible (or can't be used with another device).*
  In this case, the problem is reported by all beans that share some timing hardware. Due to dependencies between used parts of the timer, there might be necessary to adjust the values of the shared elements (like prescalers) to the same value.
  For example, if two TimerInt beans are using two channels of one timer and all timer channels are connected to one common prescaler, it is not possible to set the values that would require a different prescaler values. In

this case it is useful to manually adjust the prescaler values of all beans to the same value (switch to Expert view mode and adjust the property in the Bean Inspector window). Structure of the current target CPU timing model can be viewed by using the CPU Timing Model window.

- *The Runtime setting from interval is selected and it is not possible to set the values.*
  The required run-time settings are outside the range of one prescaler. This is a limitation of this mode of runtime setting. Please see the text below in this chapter.

Run-time Timing Settings Limitation

Some beans allow to change the timing at run-time by switching among several predefined values or by setting a value from given interval. If the setting from an interval is selected, the Processor Expert allows to configure the limits of this interval only within a range of one prescaler. This means that there has to be possible to set any value in the interval using the same prescaler value.

Speed Modes

Processor Expert provides three speed modes that are generalization of all the possible CPU clock speed modes used for power-saving that supported by most of the modern microcontrollers. See chapter *3.2.2.2 Speed Modes Support* for details.

3.3.5. Creating User Bean Templates

If you frequently use a bean with the same specific settings, you may save the bean with its settings as a template. This template is displayed in the Bean selector under given name, **behaves as a normal bean** and could be added to any project. The template has the same properties as the original bean. The values of the properties are preset in the template and could be marked as read only.
In this section, we will show how to create a bean template and save it.

How to Create and Save Templates.

Click the **right mouse** button on the selected bean icon on the Project panel in order to display the Bean pop-up menu.

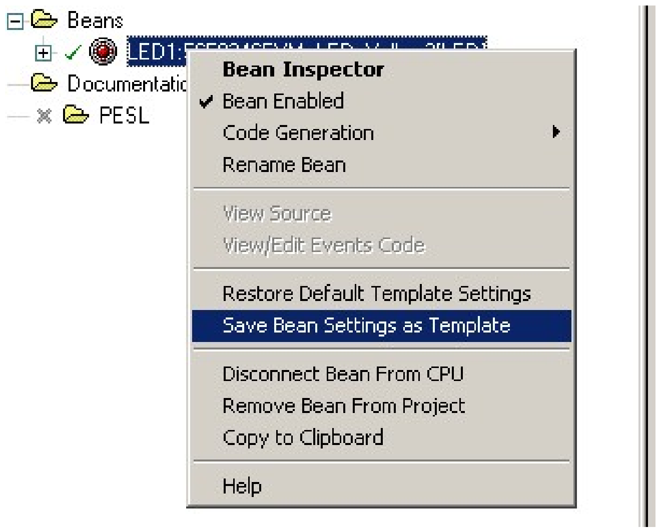


Figure 3.5 - Bean Pop-up Menu

Select the **Save bean settings as template** item to open the Bean Template dialog window which allows to create and save the template.

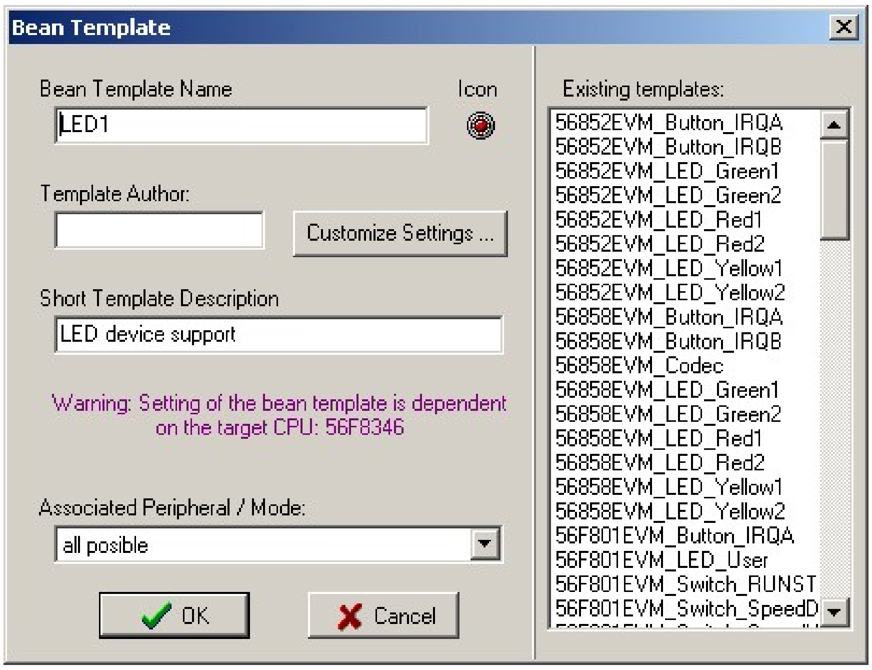


Figure 3.6 - Bean Template Dialog

Dialog window has the following parts:

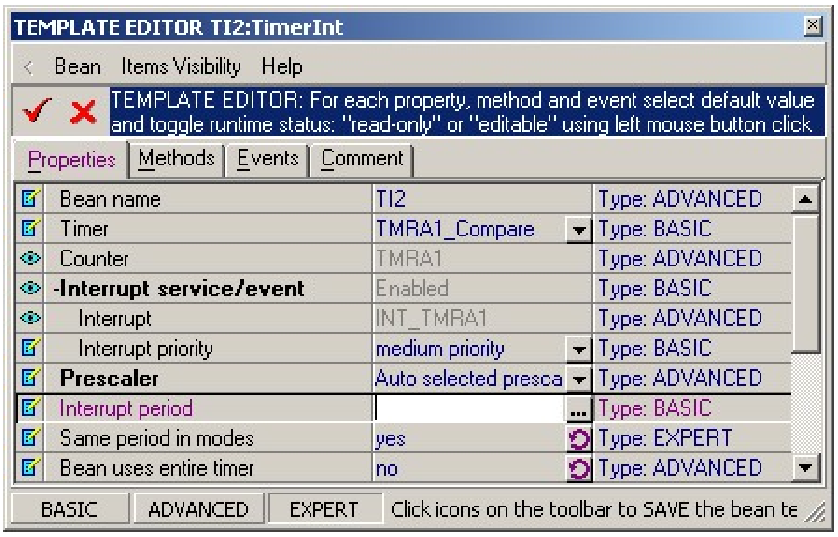
- **Bean template name:** Name of the new template
- **Icon** Icon representing the template. Click the icon to select a different icon.
- **Template author:** Your name, name of the firm etc...
- **Customize settings...** Button invokes the Template Editor which allows to change the settings of the bean. These settings will be saved as a template. (See the following section 'Customizing Bean Template Settings' for details).
- **Short template description:** Type a short description.
- User gets on of the messages: "Setting of the bean template is not dependent on the target CPU." or "Warning: Setting of the bean template is dependent on the target CPU". The second one means that the template will be shown in bean selector only for the current target CPU.
- **Associated peripheral / Mode** - *This setting is present only if it is meaningful.* If the user chooses a peripheral, the template will be shown in *OnChipPeripheral* mode only for this peripheral.
- **Existing templates** Shows list of the existing templates for the bean. Clicking on the item in the list will load the templates settings into the input fields.

After clicking the **OK** button, the bean template is saved and automatically added to the Beans Selector tree.

Customizing Bean Template Settings

If you click **Customize settings...** button in the Bean Template dialog the Template Editor dialog window is shown and you can make the following changes into the template:

- set default values of properties,
- set default values of methods or events (whether it has be generated or not generated),
- rename methods (by double clicking on method name),
- set feature of properties, methods or events as *Read Only* (the user cannot edit the default values for properties or change features like whether to generate or not to generate methods or events) or *Changeable* (user can edit default values of properties or change features generate/not generate of methods or events) by double clicking in the first column in Bean Inspector. The icon shows the status:
  - 👁 - feature is ReadOnly.
  - ✎ - feature is Changeable.
- set level of visibility of properties, methods or events by repeatedly double clicking in last column in Bean Inspector. When you use the template you may change the level of visibility in the View menu in the Bean Inspector (See chapter *2.5.3 Bean Inspector* for details.).
  Possible values:
  - **Type: BASIC** - property/method/event will always be visible.
  - **Type: ADVANCED** - property/method/event will be visible if it is selected in Advanced view.
  - **Type: EXPERT** - property/method/event will be visible if it is selected in Expert view.
  - **Type: @ HIDDEN @** - property/method/event will never be visible.



TEMPLATE EDITOR: For each property, method and event select default value and toggle runtime status: "read-only" or "editable" using left mouse button click

✎	Bean name	TI2	Type: ADVANCED
✎	Timer	TMRA1_Compare	Type: BASIC
👁	Counter	TMRA1	Type: ADVANCED
👁	**-Interrupt service/event**	Enabled	Type: BASIC
👁	Interrupt	INT_TMRA1	Type: ADVANCED
✎	Interrupt priority	medium priority	Type: BASIC
✎	**Prescaler**	Auto selected presca	Type: ADVANCED
✎	Interrupt period		Type: BASIC
✎	Same period in modes	yes	Type: EXPERT
✎	Bean uses entire timer	no	Type: ADVANCED

Click icons on the toolbar to SAVE the bean te

Figure 3.7 - Template Editor Window

Finally, click the ✓ icon on the bean inspector's toolbar to accept the changes.

To discard the changes and return without changing a template settings close the window or click the ✗ icon.

3.3.6. Signal Names

The main purpose of signals is to allow the user to name the pins used by beans with names corresponding to his application. A signal name can be assigned to an allocated pin in the bean properties (available in ADVANCED view mode).

Processor Expert automatically generates a document *{projectname}_SIGNALS.txt* or *{projectname}_SIGNALS.doc* containing a list of relationship between defined signals and corresponding pins and vice versa. There is an additional signal direction information added next to each signal name and pin number information next to each pin name. This document can be found in the *Documentation* folder of the Project Panel.

Sample of generated signals documentation:

```
=================================================================
 SIGNAL LIST
-----------------------------------------------------------------
SIGNAL-NAME [DIR]         => PIN-NAME [PIN-NUMBER]
-----------------------------------------------------------------
LED1 [Output]             => GPIOA8_A0 [138]
LED2 [Output]             => GPIOA9_A1 [10]
Sensor [Input]            => GPIOC5_TA1_PHASEB0 [140]
TestPin [I/O]             => GPIOE0_TxD0 [4]
Timer [Output]            => GPIOC4_TA0_PHASEA0 [139]
=================================================================


=================================================================
 PIN LIST
-----------------------------------------------------------------
PIN-NAME [PIN-NUM]        => SIGNAL-NAME [DIRECTION]
-----------------------------------------------------------------
GPIOA8_A0 [138]           => LED1 [Output]
GPIOA9_A1 [10]            => LED2 [Output]
GPIOC4_TA0_PHASEA0 [139] => Timer [Output]
GPIOC5_TA1_PHASEB0 [140] => Sensor [Input]
GPIOE0_TxD0 [4]           => TestPin [I/O]
=================================================================
```

3.3.7. Bean Inheritance and Bean Sharing

Basic Terms

- Ancestor is a bean that is inherited (used) by another bean.
- Descendant is a new bean that inherits (uses) another bean(s).
- Shared Ancestor is a bean that can be used and shared by multiple beans.

Inheritance

Inheritance means that an ancestor bean is used only by the descendant bean. Inheritance is supported in order to allow beans to access peripherals by hardware-independent interface of the ancestor beans.
For example, a bean that emulates a simple I2C transmitter may inherit two BitIO beans for generation of an output signal.

On several complex beans (for example some MPC5500 Peripheral Initialization beans) inheritance is used to separate bean settings into several logical parts, for example settings of channel is inherited in the bean with settings of the main peripheral module.

Settings in Processor Expert

The Descendant bean contains a property which allows selecting an ancestor bean from a predefined list of templates. The bean is created after selection of an appropriate template name (or bean name) from the list of the templates fitting the specified interface. Any previously used ancestor bean is discarded.

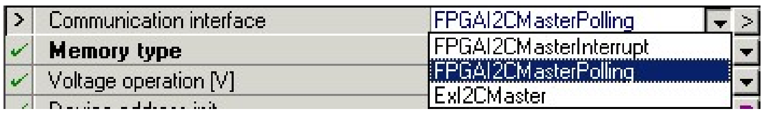

Communication interface: FPGAI2CMasterPolling
Memory type
Voltage operation [V]
FPGAI2CMasterInterrupt
FPGAI2CMasterPolling
ExI2CMaster

Figure 3.8 - Example of selecting from available ancestor beans.

Press the button to edit properties, methods or events of a selected ancestor bean in the *Bean Inspector*.
Processor Expert allows the user to select from several ancestors that implement a required interface and are registered by the descendant bean.
The ancestor bean is displayed under its descendant in the project structure tree in the project panel.

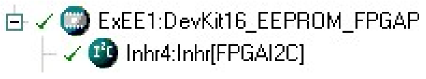

ExEE1:DevKit16_EEPROM_FPGAP
Inhr4:Inhr[FPGAI2C]

Figure 3.9 - Example of ancestor and descendant beans in the project panel tree.

An ancestor bean requires a list of methods and events (interface), which must be implemented by an ancestor bean. The error is shown if the ancestor bean does not implement any of them (for example if the settings of the descendant bean do not allow it to generate this method).

Bean Sharing

Bean sharing allows the user to cause several beans to use capability of one bean with the way similar to inheritance. This feature allows sharing of its resources and its drivers with other beans.
For example, beans may share an I2C bean for communication with peripherals connected to the I2C bus.

Settings in Processor Expert

A shared ancestor bean contains a property which allows the user to select existing or create a new shared ancestor bean. The ancestor bean is included in the project tree as are the other beans. The ancestor bean may be used with the descendant bean only if it was created from a template registered in the descendant bean or if the bean type is registered in the descendant bean. It's recommended that you always create a shared ancestor bean through a descendant bean.
Press the ▼ button to select an existing shared ancestor bean from the current project. Press the ... button to select an existing or create a new ancestor bean using the Bean Wizard (see below).

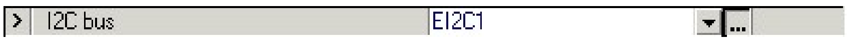
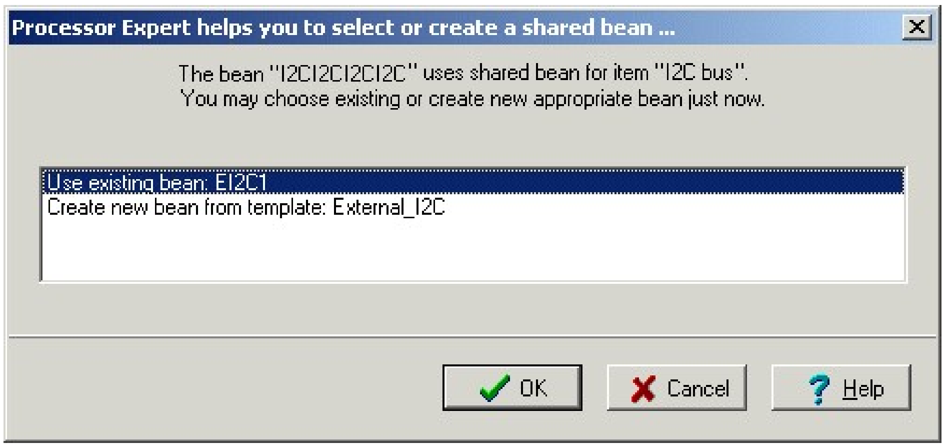


Figure 3.10 - Example of popup menu for creating a new shared ancestor bean.

Selection/Creation Wizard

When a bean with a link to a shared ancestor bean is added to the project, the following "Selecting/Creating Wizard" appears. This wizard helps you to select or create the shared ancestor bean quickly.



Processor Expert helps you to select or create a shared bean ...

The bean "I2CI2CI2CI2C" uses shared bean for item "I2C bus".
You may choose existing or create new appropriate bean just now.

Use existing bean: EI2C1
Create new bean from template: External_I2C

Figure 3.11 - Example of popup menu for creating new shared ancestor bean.

Run-time Resources Allocation

Processor Expert (generated code) does not check the usage of shared resources/code. It's up to the user to use the correct run-time resources allocation of a shared ancestor bean. Usually it is not possible for a shared ancestor bean to be used simultaneously by several beans.

3.3.8. Pin Sharing

Pin sharing is the way, how **multiple beans can use one pin of the CPU**. Processor Expert supports the following ways of pin sharing:

InputPin bean

This bean supports reading of input pin signal without any previous pin initialization. The feature can be used only if the pin (port) contains RAW DATA register, which allows reading the input pin signal in any settings of related peripherals. See bean documentation for more details. Note: InputPin bean is included in the installation of Processor Expert only if any target CPU supports the feature mentioned above.

Design-Time and Run-Time Sharing

There is also possible to select pin sharing for the pin in the bean inspector, which is shared with another bean (denoted "main" bean). In this case the pin must be allocated and initialized by the main bean. The shared bean provides CPU specific verification testing if pin sharing is supported on target CPU with the main bean and decides, if the sharing can be set in the initialization (design- time sharing), or if it is necessary to invoke a method to select switch between main and shared bean (run-time sharing). It's allowed to set sharing of one pin for several beans as well.

Pin sharing is advanced usage of the CPU peripherals and should be done only by skilled users. Pin sharing allows advanced usage of the pins even on small CPU packages and allows application-specific usage of the pins.

Pin sharing can be set in the bean inspector. The bean inspector must be switched into **EXPERT mode**, and then the pin sharing button must be switched down. See picture, button at the right side of the second column.

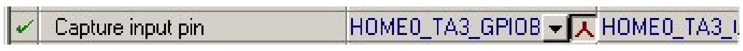

✓ Capture input pin HOME0_TA3_GPIOB HOME0_TA3_(

Figure 3.12 - Pin Sharing Button

Design-time sharing does not need any user action in run-time. Some CPUs support usage of the pin by more peripherals. For example input pin can be used as an input for capture and at the same time as an input for another counter (EventCounter). Design-time sharing is indicated by single-color icon on the button

Run-time sharing means, that it's necessary to invoke bean method ConnectPin to connect bean to the shared pin. And it's necessary to invoke main bean method to connect pin back to main bean. In fact the bean can usually operate simultaneously, but they have no connection to the shared pins unless ConnectPin method is executed. Run-time sharing is indicated by three-color icon on the button

If more beans share one pin, it's better to invoke method ConnectPin before any bean usage. It's allowed to invoke this method also during design-time sharing - the method has no effect in this case. The method ConnectPin is not generated by default and it must be turned on manually.

Shared pins are presented in the Target CPU view as well. The bean to pin connection line is red.

3.3.9. Import 56800/E Project From Quick-Start

Notice: Import From Quick-Start is supported only in the version for 56800/E.
Imports a Quick-Start project for 56800/E derivatives and creates appropriate Processor Expert project with corresponding CPU bean and Peripheral Initialization beans. Settings of the Quick-Start project are imported from the file *AppConfig.h* and these settings are applied to PE beans.

The settings are converted automatically, the only user input is to select the path to the *AppConfig.h* file. Imported beans are added into a new PE configuration, the name of this configuration is based on the path to the file AppConfig.h. The beans currently present in the project are deleted (if there are any) and all PE configurations are removed.
Option **Project Options | Generate manifest constants** for all bean modules is turned on and PESL support is also enabled.

CPU bean contains necessary settings for ROM-RAM copy routines. After the CPU bean is imported these properties should be set according to the user application intentions. These properties can be found in the CPU Build options tab. There are following properties:

- xROM-xRAM mode (should be used when xROM-xRAM copy routine is required, typically when an application is placed in the internal FLASH)
- pROM-xRAM mode (should be used when pROM-xRAM copy routine is required, typically when an application is placed in the internal FLASH)

Peripheral Initialization beans provide after-reset initialization of the corresponding CPUs on-chip peripherals and also provide the **Init** method to apply these settings from a user code. User can use the low-level techniques to write his/her code.

Limitations

- Processor Expert beans support only valid peripheral settings. All invalid (not supported on the target CPU) settings of the Quick-Start project are ignored.
- Symbols from AppConfig.h are not preserved. Symbols from CH file (manifest constants) may be used instead.

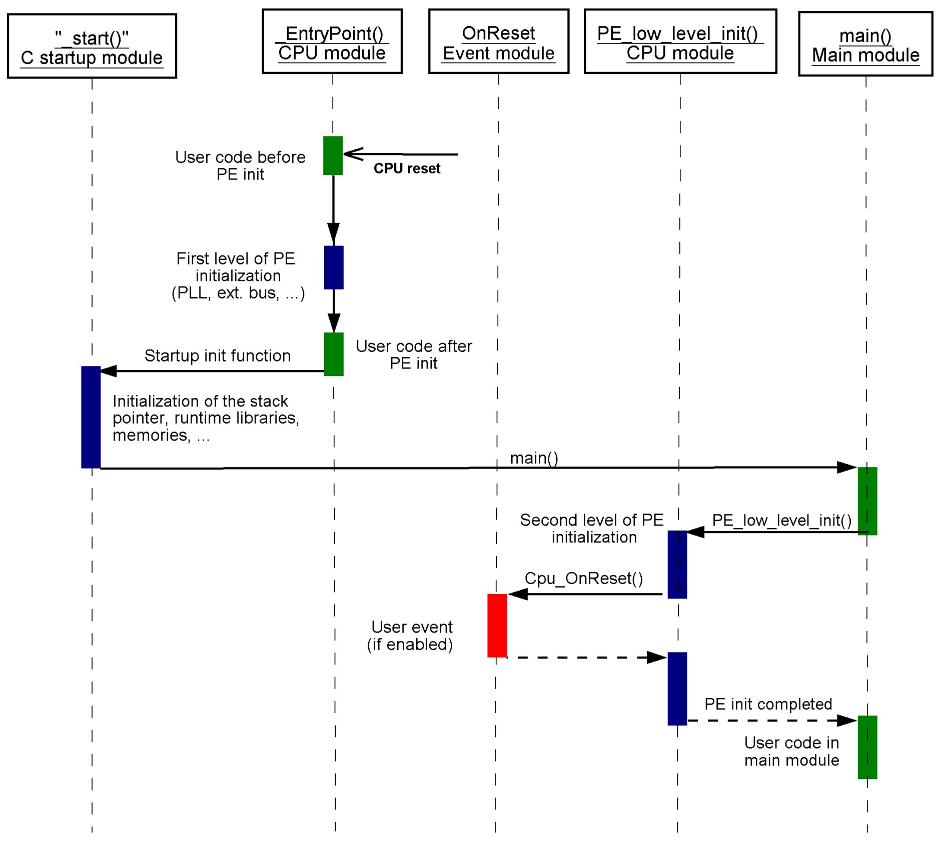
3.4. Implementation Details

This chapter contains implementation details for Embedded Beans and Processor Expert generated code. The following subchapters concerns:

- Reset Scenario With Processor Expert
- Version Specific Information for 56800/E
- Version Specific Information for HC(S)08
- Version Specific Information for HCS12 and HCS12X

Another implementation-specific information can be found on individual bean documentation pages.

3.4.1. Reset Scenario With Processor Expert



Legend:

🟩 User's code

🟦 Initialization code generated by Processor Expert

🟥 User application code in event module

◄—— Call of a method or occurence of an event

- - ► Return from method, event or interrupt

<—— Interrupt

Figure 3.13 - Reset sequence diagram

Figure 1 describes a typical initialization scenario with Processor Expert after the cpu reset.

_EntryPoint function

The *_EntryPoint()* function is called as the first function after the reset. This function is defined in the cpu module, usually *Cpu.c*, and provides necessary system initialization like PLL, external bus, etc...
Sometimes it is necessary to do some special user initialization immediately after the cpu reset. Processor Expert provides a possibility to **insert user code into the *_EntryPoint()* function**. There is a User Initialization property in the build options tab of a CPU bean inspector defined for this purpose. See chapter *2.5.3 Bean Inspector* for details.

C startup module

The C startup module is called at the end of the *_EntryPoint()* function. The C startup module provides necessary initialization of the stack pointer, runtime libraries, etc... At the end of the C startup module the *main()* function is called.

PE_low_level_init()

There is a second level of Processor Expert initialization *PE_low_level_init()* called at the beginning of the *main()* function. *PE_low_level_init()* function provides initialization of all beans in project and it is necessary for proper functionality of the Processor Expert project.

OnReset event

The user can write his/her own code that will be invoked from the PE_low_level_init() function after Processor Expert internal initialization before the initialization of individual beans. Thus, the user should expect that peripherals are not completely initialized yet. This event can be enabled/disabled in the CPU bean inspector's events page.

3.4.2. Version Specific Information for 56800/E

Priority of interrupts and events

Please see details and version specific information in the chapter 3.3.1.2 Processor Expert Priority System.

Chaining of timer channels

The timer channels can be chained. Chaining of 16-bit counters it is supported by accommodating counts up to 64bits. The chained channels can be selected by a "Timer" property. For example, if 32-bit counts are required for the FreeCntr bean, it is possible to set the "Timer" property of the bean by selecting the TMRA01_Compare or TMRA01_Free values. These counters are not "standalone" 32- bit HW counters, but rather two chained 16-bit counters.

Notice: Only chaining of the channels 0-1,2-3 and 0-1-2-3 is available. Another possible chains can be created with using Init_TMR beans.

Capture bean

Once the capture is triggered, the capture register cannot be overwritten until the 'Input edge' flag is enabled again. This is provided different way depending on the Interrupt service settings and OnCapture Event usage. The following cases can occur:

- Interrupt service is disabled. Once a capture event occurs, no further updating of the capture register will occur until the method GetCaptureValue is used (the 'Input edge' flag is enabled in this method).
- Interrupt service is enabled and event OnCapture is disabled. The 'Input edge' flag is cleared immediately after the interrupt occurs. Content of the capture register can be updated immediately with any input active transition.
- Interrupt service is enabled and event OnCapture is enabled. It is recommended to use the method GetCaptureValue within OnCapture event. Content of the capture register is protected against the change until the end of OnCapture event only.

TimeDate bean

It is recommended to set a resolution to multiples of 10 ms (resolution of the time provided by the GetTime/SetTime methods). It should be 10ms or more. Smaller values are unnecessarily overloading the system.

PulseAccumulator bean

This bean is generally used to count pulses (events) generated on external inputs. Thus, the primary and secondary input can only be a physical pins of the device. The primary input is required to be an internal clock, the Init_TMR bean has to be used instead.

WatchDog bean

The interrupt service routine for the vector INT_COPReset is generated only if the OnWatchDog event is used. Otherwise the INT_COPreset entry in the interrupt vector table contains only the call of the _EntryPoint, which is the same as the INT_Reset service routine. The user can find out the cause of the reset by using a CPU bean method GetResetSource.

FreescaleCAN bean

This bean can encapsulate FlexCAN device or MSCAN12 device.

- **FlexCAN device**
  The FlexCAN device receives self-transmitted frames if there exist a matching receive MB. FlexCAN module is implemented on 56F83xx derivatives. Message buffers should be configured as receive or transmit using the FreescaleCAN bean's settings.
- **MSCAN12 device**
  When interrupt mode is enabled, received frames should be read in the OnFullRxBuffer event to avoid message buffer lock/unlock problems.

AsynchroSerial, SynchroMaster, SynchroSlave, FreescaleSSI beans

When the bean is configured in DMA mode then Send/Receive routines use a user buffer that is passed as a parameter to these methods. User should avoid changing a buffer content during receive/transmit process.

IntFlash beans

If the Save write method is used (property Write method), the Save buffer (buffer for saving data from the sector which has to be erased) is implemented by bean in data RAM.

If the Virtual page feature is used (property Virtual page), the page buffer is implemented by bean in data RAM.

The basic addressing mode of IntFLASH bean methods is a 16-bit word. It is used by most of the memory access methods. Only SetByteFlash, GetByteFlash, SetBytePage, GetBytePage and SetBlockFlash, GetBlockFlash methods use a byte addressing mode. An address of the byte location is an address according to a 16-bit word location multiplied by 2 and then the even/odd bytes are discriminated by LSB: 0 for even byte, 1 for odd byte.

PE does not check if the memory mode selected in the CPU bean corresponds to the current CodeWarrior target. Thus it is needed to take care to the memory mode selection especially if the program and boot flash memory is served by the IntFLASH bean (if the program and boot flash memory has to be served by the bean, then one of the internal memory targets has to be selected).

- **56F83xx, 56F81xx, 56F80xx** derivatives:
  If the project contain both IntFLASH beans (one for each memory space), then none of the beans could be disabled in High speed mode.

  If a programming/erasing operation is started by bean and it is configured not to wait until the end of the operation (property Wait enabled in init., method SetWait), then calling of a programming/erasing method of the other bean is not allowed before the end of the programming/erasing operation of the first bean (ERR_BUSY is returned).

- **56F80x, 56F82x** derivatives:
  Internal flash has not protection feature, so the SetProtection and SetGlobalProtection methods are not implemented.

  If the bean is configured not to wait until the end of the programming/erasing operation (property Wait enabled in init., method SetWait), the FinishProcess method has to be called after the end of the operation.

  Since the flash device does not support erase verification feature, the EraseVerify method is implemented by software routine. Thus it gets more time to verify the flash memory than this method is implemented by hardware module (all parts of the flash memory have to be read).

3.4.3. Version Specific Information for HC(S)08

The ROM, Z_RAM and RAM ranges depend on the target CPU. It is recommended to increase the stack size if some standard libraries are used.

For the detailed information on debugging HC08 application using **MON8** interface please follow to the chapter 3.4.3.2 Debugging on HC08 Using MON8.

Beans' implementation details:

- **All the beans**:
  - *Interrupt priority* - the value of this property is ignored, since the HC08 has no HW support for setting interrupt priorities.
  - *Event priority* - the value of this property can be only "0" (interrupts disabled) or "1" (interrupts enabled), since the HC08 has no HW support for setting interrupt priorities.

  For details on priority settings for interrupts and event please see also the chapter 3.3.1.2 Processor Expert Priority System

- **CPU**:
  - *Speed Mode* selection (CPU methods *SetHighSpeed*, *SetLowSpeed*, *SetSlowSpeed*): if CPU clock-dependent beans are used then signals generated from such internal peripherals may be corrupted at the moment of the speed mode selection (if function of clocked devices is enabled). Handling of such a situation may be done via events **BeforeNewSpeed** and **AfterNewSpeed**.
  - *Interrupt vector table in ROM is placed* at the default address in the ROM or in the Flash.
    If the *interrupt vector table in RAM* is selected then it is generated the table in RAM and special redirection code to ROM. This code transfers program control to the selected address according the table in RAM. You can use CPU methods SetIntVect to set the address of interrupt service routine.
    Note: you cannot change the interrupt vector that is allocated by any bean in your project. It is recommended to select the event OnSWI together with this option to minimize size of generated code.

- **PPG**: The PPG beans always allocates the whole timer. Although it would be possible to share the selected timer between 2 PPG beans, it would be impossible to set the PPG period for these two beans separately. (More information about this bean can be found in chapter Timers)
- **PWM**: In contrast to the PPG beans, it is possible for PWM beans to share the selected timer, since they do not have the *SetPeriod* method. (More information about this bean can be found in chapter Timers)
- **EventCntr16**: Since the timer overflow flag is set when the timer reaches a value of 65535, the maximum number of events that can be counted by this bean is limited to 65534 (value of 65535 is marked as invalid as the method *GetNumEvents* returns the ERR_OVERFLOW value as its result.) (More information about this bean can be found in chapter Timers)
- **TimeDate**: It is recommended to make a setting close to 10 ms (resolution provided by GetTime/SetTime methods). Smaller values unnecessarily overload the system.
- **WatchDog**: When the Watchdog bean is added to the project, it is automatically enabled. The enabling code is placed in the CPU initialization code.
  Note: Watchdog is enabled by a write to the configuration register. This register can be written only once after CPU reset. Since the register also contains other bits, that are written during the CPU initialization, the watchdog must be enabled when CPU is initialized. The property "CPU clock/speed selection" has no effect because the COP timer clock source is CGMXCLK.
- **AsynchroSerial**:
  - Timing setting 'values from list' enables to select various values denoted by changes of the prescaler most tightly coupled with the UART.
  - If a software handshake is used for extremely high baud-rates it may happen that no overruns appear and transmitted characters get lost
- **AsynchroMaster**: the same as AsynchroSerial
- **AsynchroSlave**: the same as AsynchroMaster.
- **SynchroMaster**: Because of the disability of an SPI device (configured as Master) caused by a mode fault, the mode fault automatically disables the bean (inside interrupt service) if interrupt service is enabled. If the interrupt service disabled and a mode fault occurs, the bean will be disabled at the beginning of RecvChar method.
- **SynchroSlave**: A mode fault doesn't disable an SPI device (configured as Slave), therefore it doesn't disable the bean.
  If a mode fault error occurs, software can abort the SPI transmission by disabling and enabling of the device ('Enable' and 'Disable' methods). When Clock edge property = "falling edge", Shift clock idle polarity property = "Low" or Clock edge property = "rising edge", Shift clock idle polarity property = "High" the SS pin of the slave SPI module must be set to logic 1 between bytes. The falling edge of SS indicates the beginning of the transmission. This causes the SPI to leave its idle state and begin driving the MISO pin with the MSB of its data. Once the transmission begins, no new data is allowed into the shift register from the data register. Therefore, the slave data register must be loaded with the desired transmit data before the falling edge of SS.
- **BitIO, BitsIO, ByteIO, Byte2IO, Byte3IO, Byte4IO**:
  The *GetVal* and *GetDir* methods are always implemented as macros. *Optimization for* property (BitIO, BitsIO) doesn't influence the generated code.
- **WordIO, LongIO**:
  These beans could not be implemented on Freescale HC08 - this CPU has no instructions for 16-bit and 32-bit access into the I/O space.
- **ADC, ADconverter**:
  There are the following restrictions in Processor Expert:

  - Clock input of A/D clock generator cannot be changed in runtime.
  - The voltage levels supplied from internal reference node cannot be measured.

  A conversion time in the 'Conversion time' dialog is calculated for the worse case, that is 17 cycles per one conversion.
- **ADfast**:
  There are the following restrictions in Processor Expert:
  - Clock input of A/D clock generator cannot be changed in runtime.
  - The voltage levels supplied from internal reference node cannot be measured.
  - ADC device doesn't support continuous mode through adjoining channels. Therefore the bean does measurement in single mode.

  A conversion time in the 'Conversion time' dialog is calculated for the worse case, that is 17 cycles per one conversion.
- **ExtInt**:
  If a pin other than IRQ (IRQ1) is set in this bean, setting of the 'Pull resistor' property affects only disable state of the device (bean). If the device (bean) is enabled, the pull-up resistor is always connected to the pin.
- **KBI**:
  - Setting of the 'Pull resistor' property affects only the disabled state of the device (bean). If device (bean) is enabled, the pull-up resistors are always connected to the used pins.
  - Only one bean can be used in PE project.
- **IntEEPROM**:
  A bean expects that all security options of EEPROM are disabled. If some security option is enabled methods performing write operation (such as SetByte) can return an error.
- For details on sharing and usage of the **high-level timer beans** please see the chapter 3.4.3.1 HC(S)08 Timers Usage and Sharing.

3.4.3.1. HC(S)08 Timers Usage and Sharing

The HC(S)08 family provides two main groups of timer devices.

Simple Timer Peripherals

These devices are simple counters that do not contain multiple individually configurable channels and usually do not allow to control any pins. These devices cannot be shared (i.e. used by multiple beans).

The following devices are members of this group:

- **HC08:** PIT (TIM on some derivatives), TBM, PWU, RTC
- **HCS08:** RTI, CMT, MTIM

These devices are usually listed in Processor Expert under their original names without any extensions. These devices can be used by the following high-level beans: **TimerInt, RTIshared, TimerOut and FreeCntr8,16,32.** The MTIM peripheral can additionally be used in event counter beans (EventCntr8,16,32). All peripherals from this group are also supported by the Peripheral Initialization Beans (for details please see the chapter 3.2.1.1 Bean Categories).

Complex Timer Peripherals

These timer peripherals provide multiple channels and allow several modes of operation.

The following devices are members of this group:

- **HC08:** TIM (Timer Interface Modules)
- **HCS08:** TPM (Timer/PWM modules)

Processor Expert shows each of these timer peripherals as multiple devices that can be used by the beans. The name of such device (shown in the peripheral selection list) consists of the peripheral and a suffix specifying the part of the peripheral or it's specific function. These named devices represent the whole peripheral or parts of the peripheral set to work in a specific mode. Using only a part of the timer allows to share the timer by multiple beans. To display a list of all timer devices and their usage information, open the Peripheral Usage window and switch to tab Timers (see details in the chapter 2.15 Peripherals Usage).

The following devices are usually defined for the complex timer peripherals:
(the examples are for MC68HC908AZ60 CPU)

- **TIMx (e.g. TIMA)** - the whole timer including all channels and control registers. (Sometimes the 'x' is omitted if there is only one such timer on the chip). This device blocks all other devices defined for this timer.
- **TIMxy (e.g. TIMA0)** - channel 'y' of the timer 'x'. (Sometimes the 'x' is omitted if there is only one timer on the chip.)
- **TIMx_PPG (e.g. TIMA_PPG)** - the whole timer in a programmable pulse generation mode.
- **TIMxPP (e.g. TIMAPP)** - modulo register of the timer in a programmable pulse generation mode.
- **TIMxyPPG (e.g. TIMA0PPG)** - channel 'y' of the timer 'x' in the programmable pulse generation mode. (Sometimes the 'x' is omitted if there is only one timer on the chip.)
- **TIMxfree (e.g. TIMAfree)** - this device represents the overflow flag and interrupt capabilities of the counter. The range is of the counter is not limited and it is determined by the size of the counter register so the timing is controlled only by the prescaler selection.

Note: Even though the multiple devices defined for a timer are configured independently, they can be mutually dependent. For example, they share one common prescaler. Processor Expert instantly checks beans configuration and only valid combinations are allowed. See chapter 3.3.4 Timing Settings for details.

Using Complex Timers in High Level Beans

This chapter explains the options of usage of the complex timers in the high-level beans allowing to benefit from the advanced features of these beans. All peripherals from this group are also supported by the Peripheral Initialization Beans (for details please see the chapter 3.2.1.1 Bean Categories).

The following table shows a list of the timer beans and PE devices that can be used in the beans as rows. The columns show the state of all devices defined for the timer for conditions determined by rows.

Bean	Selected device(s)	PE devices defined for the timer					
		TIMx	TIMxy	TIMxfree	TIMx_PPG	TIMxPP	TIMxyPPG
PWM	TIMxy	Blocked	Used 1 channel (2 in buffered mode) Others free	Blocked	Blocked	Blocked	Blocked
PPG	TIMxPP TIMxyPPG	Blocked	All channels blocked	Blocked	Used	Used	Used
TimerOut	TIMxy	Blocked	Used 1 channel (2 in buffered mode) Others free	Free	Blocked	Blocked	Blocked
TimerInt RTIShared TimeDate FreeCntr8 FreeCntr6 FreeCntr32	TIMxy	Blocked	Used 1 channel Others free	Free	Blocked	Blocked	Blocked
	TIMxfree	Blocked	All channels free	Used	Blocked	Blocked	Blocked
EventCntr8 EventCntr16 EventCntr32	TIMx	Used	All channels blocked	Blocked	Blocked	Blocked	Blocked
Capture	TIMxy	Blocked	Used 1 channel Others free	Free	Blocked	Blocked	Blocked

Table legend:

- Blocked - the device might not be directly used by the bean but it cannot be used (shared) by other beans because it would disrupt the beans' function.
- Used - the device is required and used by the bean.
- Free - the device is not used nor blocked by the bean so it can be used by another bean.

How to Use the Table

The table allows to easily find out which beans (in which setup) can share the timer peripheral. The following rule determines the condition necessary for sharing : **When we take the rows of table corresponding to the beans and their configurations we want to use, every column containing "Used" value must contain "Free" in all other rows (it cannot be used or blocked). In case of individual channels there has to be enough channels for all beans.** Please notice that if a bean allocates some channels, it is possible to share the timer among several beans of the same type (e.g. TimerInt using the TIMxy device).

PWM Sharing Limitation

There are some limitation for the PWM bean, if it shares the timer peripheral with other devices. The PWM in this case uses the whole range of the counter (i.e. the modulo register is not used) so the period values are limited to the value(s) determined by the prescaler value.

Example

The 68HC908AZ60 contains two timer modules TIMA and TIMB. Each one of these modules is based on a 16-bit *counter that can operate as a free-running counter or a modulo-up counter. TIMB module has 2 channels and 2 related input/output pins. Can we use the TIMB peripheral for PWM output and input capture beans at once?*

Timer Module B (TIMB) is supported in PE by the following devices: TIMB, TIMB0, TIMB1, TIMBfree, TIMB_PPG, TIMBPP, TIMB0PPG, TIMB1PPG.
It follows from the table in this chapter that:

- For the PWM bean (in non-buffered mode) we use only one channel (e.g. **TIMB0**). According to the columns, the bean will use one channel and TIMBfree device, other channels will stay free, all other devices will be blocked.
- The Capture bean, according to the columns, uses only one channel. Such channel is available: **TIMB1**.
- Sharing of the TIMB peripheral by the PWM and Capture bean is possible. There is no remaining free device on the timer peripheral.

Because the channels of this timer are sharing one prescaler, it is necessary to configure the same prescaler value for both PWM bean and Capture bean.

3.4.3.2. Debugging on HC08 Using MON8

Every member of the HC08 microcontroller family is equipped with a basic support for in-system programming and debugging (MON8, for details see datasheet of a HC08 processor). The microcontroller can work in two modes - normal mode and monitor mode.

In the monitor mode the microcontroller can accept couple of commands over the single wire interface. The commands allow to read/write the memory and run a code. In combination with Break module a simple debugging system can be built (e.g. ICS boards, P&E Multilink or various custom designs).

There are few issues that results from the characteristics of the the MON8 system:

- To achieve a standard communication speed (19200, 9600, 4800 bauds) a specific oscillator frequency must be used (usually 9.83 or 4.915MHz). Suitable source of processor clock is usually part of the debugging system. The user must set the same clock frequency in the CPU bean of his project to ensure that the timing of components will be correct. Care must be taken when using PLL and SpeedModes. Change of the operating frequency of the target processor can result in lost of communication with the target system.
- Some processor models allows to by-pass internal divider-by-2, which effectively doubles the bus clock of the processor. The bypass is selected by logic state of selected input pin (e.g. PTC3) during processor reset. The user must set appropriate property in the CPU bean to reflect actual state of the pin.
- One I/O pin (e.g. PTA0) is used for communication with the host computer, therefore it can't be used as a general I/O pin.
- In some configuration of the debugging system the IRQ pin can be also used to control the target board, therefore it can't be used in user application.

Capturing unused interrupts

The debugging system based on MON8 allows only one breakpoint placed in the flash memory. However, executing an SWI instruction while running is functionally equivalent to hitting a breakpoint, except that execution stops at the instruction following the SWI. The user can use this feature to actively capture unused interrupts. There are two options of capturing such interrupts :

- If the property named '*Unhandled interrupts*' located in *Build Options* tab is set to *Own handler for every*, there is an interrupt routine generated for each unhandled interrupt generated into CPU.c module. The SWI instruction can be placed in the generated routine of the interrupt that need to be caught.
- The user can also use the InterruptVector bean. In the properties of the bean select which interrupt will be monitored and set the name of the ISR function - e.g. Trap. One function can be used to capture more interrupts if property *Allow duplicate ISR names* is set to *yes*. The Trap function will contain only the SWI instruction:

```
__interrupt void Trap(void)
  {
    asm(SWI);
  }
```

3.4.4. Version Specific Information for HCS12 and HCS12X

All beans were tested with the CodeWarrior with the following compiler settings:

- Other parameters = -Onf

The ROM and RAM ranges depend on the target CPU. It is recommended to increase the stack size if some standard libraries are used.

Beans' implementation details :

- **All the beans**:
  - *Interrupt priority* and *Event priority* - Please see the version specific information details in the chapter 3.3.1.2 Processor Expert Priority System.
- **CPU**:
  - *Speed Mode* selection (CPU methods *SetHighSpeed*, *SetLowSpeed*, *SetSlowSpeed*): if CPU clock-dependent beans are used then signals generated from such an internal peripheral may be corrupted at the moment of the speed mode selection (if function of clocked devices is enabled). Handling of such a situation may be done via events **BeforeNewSpeed** and **AfterNewSpeed**.
  - *Interrupt vectors table (IVT)* is by default generated on the default addresses for of the current target CPU. However, Processor Expert offers additional configuration of the IVT:
    - **On HCS12 derivatives:**
      The placement of the IVT can be configured in the Build Options tab of the CPU Bean Inspector by changing the address of the memory area with the name **INT_VECTORS**.
      Please notice that if the IVT placement is changed, the user has to provide a full IVT on the the address defined by the CPU datasheet and the vectors allocated by Processor Expert have to be redirected into the IVT generated by PE.

If the *interrupt vector table in RAM* application option is selected then it generates the table in RAM and special redirection code to ROM. This code transfers program control to the selected address according the table in RAM. You can use the CPU method *SetIntVect* to set the address of interrupt service routine. It is recommended to select the event OnSWI together with this option to minimize the size of generated code. *Please notice that the redirection is available only for interrupt vectors not used by Embedded Beans in the current project.*

- **On HCS12X derivatives:**
  These derivatives allow to change the placement of the interrupt vectors beginning. So the Processor Expert allows to adjust both - the physical placement of vectors or the placement of the generated IVT. Please see the content of the property group *Interrupt/Reset vector table* in the group *Interrupt resource mapping* and its documentation.

- **PPG**: HW doesn't support an interrupt. Aligned Center Mode Counter counts from 0 up to the value period register and then back down to 0. If the align mode is switched to Center align mode then real lengths of Period and Starting pulse width signals will be twice as much as is being displayed in the Bean Inspector. *Note: See the Internal peripheral property group of the CPU bean for special settings.*
- **PWM**: HW doesn't support an interrupt. Aligned Center mode Counter counts from 0 up to the value period register and then back down to 0. If align mode is switched to Center align mode then the real lengths of Period and Starting pulse width signals will be twice as much as is being displayed in the Bean Inspector. *Note: See the Internal peripheral property group of the CPU bean for special settings.*
- **EventCntr8/16/32**: Functionality of this bean is a subset of the pulse accumulator. For work with hold registers, gated time mode use the PulseAccumulator bean instead of the EventCounter bean.
- **PulseAccumulator**:
  - **Method Latch**
    This method causes capture of the counter in the hold registers of all capture and pulse accumulator beans in PE project because this method is invoked for all ECT modules.

  Note: See Internal peripheral property group of the CPU bean for special settings.
- **Capture**:
  - **Method Reset** -If the counter can't be reset (is not allowed by HW or the counter is shared by more beans) this method stores the current value of the counter into a variable instead of a reset.
  - **Method GetValue** -If the counter can't be reset (is not allowed by HW or the counter is shared by more beans) this method doesn't return the value of register directly, but returns the value as a difference between the register value and the previously stored register value. This causes values that are proportional to time elapsed from the last invocation of the method Reset.
  - **Method Latch** -This method causes capture of the counter in the hold registers of all capture and pulse accumulator beans in PE project because this method is invoked for all ECT modules.
  - **Method GetHoldValue** -This method transfers the contents of the associated pulse accumulator to its hold register.

  Note: See the Internal peripheral property group of the CPU bean for special settings.
- **BitIO, BitsIO, ByteIO, Byte2IO, Byte3IO, Byte4IO**:
  The *GetVal* and *GetDir* methods are always implemented as macros.
- **LongIO:**
  This bean could not be implemented on Freescale HCS12 - this CPU has no instructions for 32-bit access into the I/O space.
- **IntEEPROM:**
  The EEPROM array is organized as rows of word (2 bytes), the EEPROM block's erase sector size is 2 rows

(2 words). Therefore it is preferable to use word aligned data for writing - methods SetWord and SetLong - with word aligned address or to use virtual page - property 'Page'. The size has to be a multiple of 4 bytes.

- **SynchroMaster:**
  The mode fault causes disability of the bean (and SPI device) automatically (inside interrupt service) if interrupt service is enabled. If the interrupt service isdisabled and a mode fault occurs, the bean will be disabled at the beginning of RecvChar method.
- **IntFlash:**
  The Virtual page - *Allocated by the user* feature and corresponding methods and events are not implemented.
- **ExtInt:**
  If XIRQ is selected, the method 'Disable' can't be generated, because it isn't supported by hardware. For pins of H, J, and P ports it is not possible to switch pull resistor (pull up/pull down) and sensitive edge (rising edge/falling edge) arbitrarily. Because of hardware limitations, pull down with falling edge and pull up with rising edge settings aren't allowed.

3.5. Code Generation

This chapter informs the user about principles and results of the Processor Expert code generation process and the correct ways and possibilities of using this code in the user application.

Please follow to the subchapters for more information.

- Code Generation
- Predefined Types, Macros and Constants
- Typical Usage of the Bean in the User Code
- User Changes in Generated Code

3.5.1. Code Generation

Processor Expert | Generate Code '{ProjectName.mcp}'

Generate Code command initiates the code generation process. During this process source code modules containing functionality of the beans contained in the project are generated. The project must be set-up correctly for successful code generation. If the generation is error-free all generated source code files are saved to the destination directory.

Processor Expert produces these files:

The existence of the files can be conditional to project or Processor Expert environment settings and their usage by the beans.

- **Bean module**
  This module with its header file is generated for every bean in the project with exception of some beans that generate only an initialization code or special source code modules. Name of this file is the same as the name of the bean.
  Header file (.h) contains definitions of all public symbols, which are implemented in the bean module and can be used in the user modules.
  The module contains implementation of all enabled methods and may also contain some subroutines for internal usage only. This module could be modified manually under certain conditions, but it is

recommended for experienced users only. See chapter *3.5.4 User Changes in Generated Code* for details.
Processor Expert also allows to track and review changes in the generated modules. See chapter *3.5.1.2 Tracking Changes in Generated Code* for details.

- **CPU module**
  The CPU module is generated according to the currently active target CPU bean. The CPU module contains additionally:
  - CPU initialization code
  - interrupt processing

- **Main module**
  The main module is generated only if it does not already exist (if it exists it is not changed). Name of this module is the same as the name of the project.
  The main module contains the **main** function, which is called after initialization of the CPU (from the CPU module). By default this function is generated empty (without any reasonable code). It is designed so that the user will write his/her own particular code here.

- **Event module**
  The event module is generated only if it does not exist. If it exists, only new events are added into the module; user written code is not changed.
  The event module contains all events selected in the beans. By default these event handler routines are generated empty (without any meaningful code). It is considered that user will write his/her own particular code here.
  Event module can also contain the generated ISRs for the beans that require a direct interrupt handling (Peripheral Initialization Beans). Generation of the ISRs is controlled by the project option **Project Options | Generate ISR**.
  Note: It is possible to change a name of the event module in ADVANCED view mode of the Bean Inspector.

- **Method list file** with description of all beans, methods and events generated from your project. The name of the file is *{projectname}.txt* or *{projectname}.doc*. This documentation can be found in the Documentation folder of the Project Panel.

- **Signal names**
  This is a simple text file *{projectname}_SIGNALS.txt* or *{projectname}_SIGNALS.doc* with a list of all used signal names. The signal name can be assigned to an allocated pin in the bean properties (available in ADVANCED view mode). This documentation can be found in the Documentation folder of the Project Panel. See chapter *3.3.6 Signal Names* for details.

- **XML documentation** containing the project information and settings of all beans in XML format. The generated file *{projectname}_Settings.xml* can be found in the Documentation folder of the Project Panel. It is updated after each successful code generation.

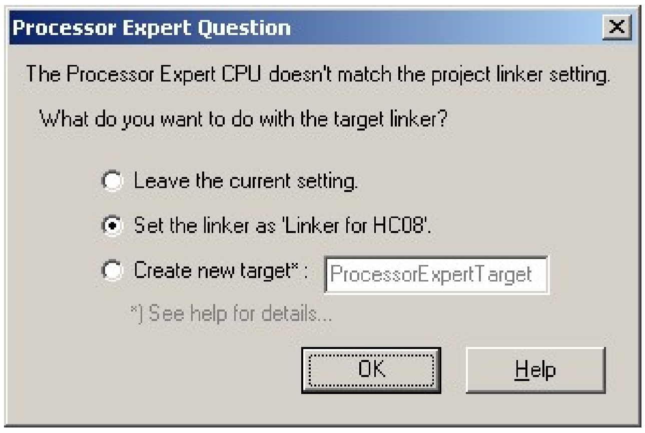
- **Shared modules** with shared code (the code which is called from several beans). Complete list of generated shared modules depends on selected CPU, language, compiler and on the current configuration of your project. Typical shared modules are:
  - **IO_Map.h**
    Control registers and bit structures names and types definitions in C language.

- **IO_Map.c**
  Control registers variable declarations in C language. This file is generated only for the HC(S)08/HC(S)12 versions.
- **Vectors.c**
  A source code of the interrupt vector table content.
- **PE_Const.h**
  Definition of the constants (speed modes, reset reasons). This file is included in every driver of the bean.
- **PE_Types.h**
  Definition of the C types (bool, byte, word, ...). This file is included in every driver of the bean.
- **PE_Error.h**
  Common error codes. This file contains definition of return error codes of bean's methods. See the generated module for detailed description of the error codes. This file is included in every driver of the bean.
- **PE_Timer**
  This file contains shared procedures for runtime support of calculations of timing constants.
- **{startupfile}.c**
  This external module (visible in the External Modules folder of the Project Panel) contains a platform specific startup code and is linked to the application. The name of the file is different for the Processor Expert versions. For datils on the use of the startupfile during the reset see chapter 3.4.1 Reset Scenario With Processor Expert
- **"PESL".h**
  PESL include file. This file can be included by the user in his/her application to use the PESL library. See chapter *3.7.1 Processor Expert System Library* for details.

See also chapter Predefined types, macros and constants.

3.5.1.1. Linker Dialog

The code generation process checks the setting of the selected target. If the selected CPU doesn't match a valid setting of the linker for the current CodeWarrior target, the code generation process displays the following dialog:


Processor Expert Question

The Processor Expert CPU doesn't match the project linker setting.

What do you want to do with the target linker?

( ) Leave the current setting.
[✓ Set the linker as 'Linker for HC08'.]
( ) Create new target* : ProcessorExpertTarget

*) See help for details...

Figure 3.14 - Linker Dialog

Following options are available:

- **Leave the current setting** - generate files to the current selected target without a change of the linker

- Second option automatically sets the linker setting according to the selected CPU, including linker settings
- **Create new target** - creates a new target and sets the linker and linker settings according to the selected CPU. Generated files will be added into the new target. You can enter a name of the target. ***Settings of this new target (i.e. entry point, libraries, access paths, target's file mappings etc.) has to be configured manually by the user to configure the target to be built properly.***

Click on the "OK" button to confirm the selection.

The "Targets" can be set for the project files in the CodeWarrior project window. To set the Target options, double click on the name of the Target listed in the window. You can also change the setting using the command **{CurrentBuildTargetName} Settings** (ALT+F7) in the Edit menu in the CodeWarrior main panel. The following picture shows the "Targets" TAB in the CodeWarrior project window:

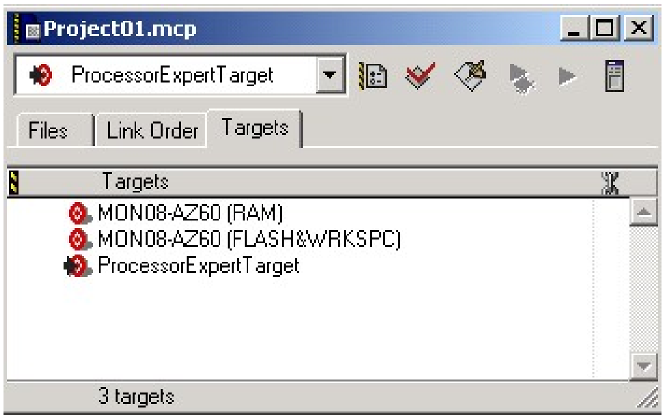
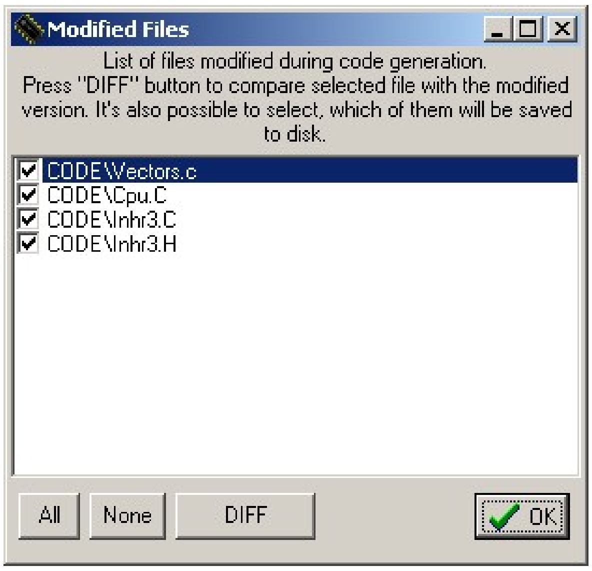


Figure 3.15 - Targets List in the CodeWarrior

3.5.1.2. Tracking Changes in Generated Code

Processor Expert allows to compare generated modules with the previously generated ones after each code generation which may prevent from unwanted changes in the bean modules. This function has to be enabled by the **Project Options | Track changes**.

The 'Modified Files' dialog appears after the successful code generation and shows the list of the files that have been modified by Processor Expert. The user can specify which files will be saved and will replace the old ones using a check marks and a buttons 'All' and 'None'.
The user can also visually compare the changes in the currently selected file by pressing a button **DIFF**. Pressing the 'OK' button will save all selected modules and replace the current ones.

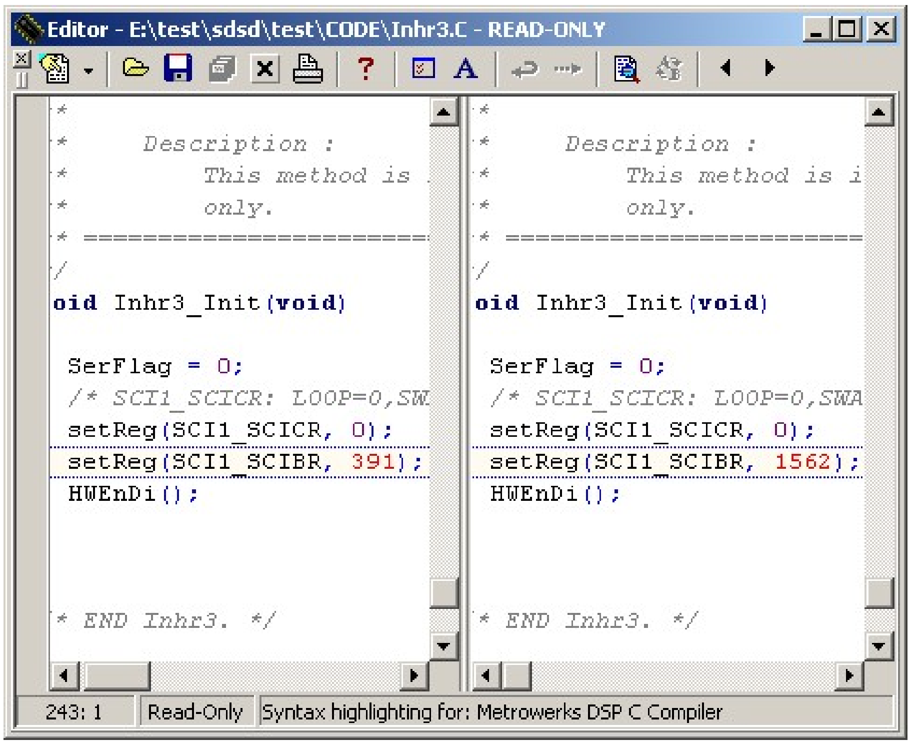


List of files modified during code generation.
Press "DIFF" button to compare selected file with the modified version. It's also possible to select, which of them will be saved to disk.

- [✓] CODE\Vectors.c
- [✓] CODE\Cpu.C
- [✓] CODE\Inhr3.C
- [✓] CODE\Inhr3.H

Figure 3.16 - List of modyfied files

If the user presses the *DIFF* button, a file editor in a comparison mode is shown. It contains highlighted parts that had been changed during the code generation. See chapter *2.16 File Editor* for details.



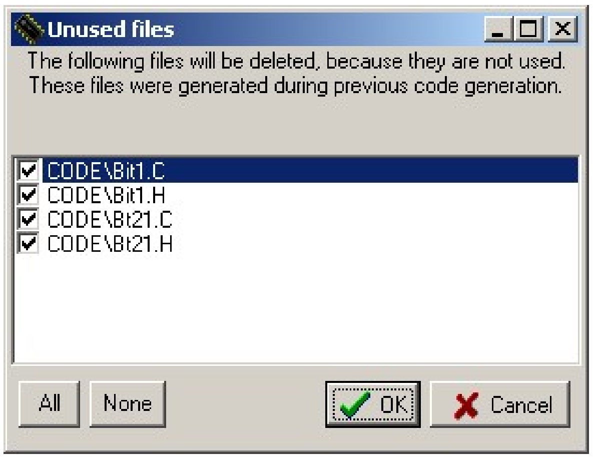
```
*
*     Description :
*         This method is
*         only.
* ========================
/
oid Inhr3_Init(void)

 SerFlag = 0;
 /* SCI1_SCICR: LOOP=0,SW
 setReg(SCI1_SCICR, 0);
 setReg(SCI1_SCIBR, 391);
 HWEnDi();


* END Inhr3. */
```

```
*
*     Description :
*         This method is i
*         only.
* ========================
/
oid Inhr3_Init(void)

 SerFlag = 0;
 /* SCI1_SCICR: LOOP=0,SWA
 setReg(SCI1_SCICR, 0);
 setReg(SCI1_SCIBR, 1562);
 HWEnDi();


* END Inhr3. */
```

243: 1 | Read-Only | Syntax highlighting for: Metrowerks DSP C Compiler

Figure 3.17 - A file before and after the generation

After the 'Modified Files' dialog, user can also review a list of automatically deleted unused files (The **Environment Options | Delete unused files** environment option has to be enabled).


Unused files

The following files will be deleted, because they are not used.
These files were generated during previous code generation.

- [✓] CODE\Bit1.C
- [✓] CODE\Bit1.H
- [✓] CODE\Bt21.C
- [✓] CODE\Bt21.H

Figure 3.18 - List of unused files

3.5.2. Predefined Types, Macros and Constants

Processor Expert generates definitions of all hardware register structures to the file **IO_Map.h.** The Processor Expert type definitions are generated to the file **PE_Types.h** containing also definitions of macros used for a peripheral register access. See chapter *3.7.2 Direct Access to Peripheral Registers* for details.

Types

Type	Description	Supported for
int8_t	8-bit signed integer	all
int16_t	16-bit signed integer	all
int32_t	32-bit signed integer	all
int64_t	32-bit signed integer	56800/E
uint8_t	8-bit unsigned integer	all
uint16_t	16-bit unsigned integer	all
uint32_t	32-bit unsigned integer	all
uint64_t	64-bit unsigned integer	56800/E
TPE_ErrCode	Error code (uint8_t)	all except MPC55xx
byte	8-bit unsigned integer (unsigned char)	all
bool	Boolean value (unsigned char) (TRUE = any non-zero value / FALSE = 0)	all
word	16-bit unsigned integer (unsigned int)	all
dword	32-bit unsigned integer (unsigned long)	all
dlong	array of two 32-bit unsigned integers (unsigned long)	all

Structure for images

```
typedef struct {          /* Black&White Image  */
```

```
    word width;             /* Image width  */
    word height;            /* Image height */
    byte *pixmap;           /* Image pixel bitmap */
    word size;              /* Image size   */
    char *name;             /* Image name   */
  } TIMAGE;
  typedef TIMAGE* PIMAGE ;  /* Pointer to image */
```

Structure for 16-bit register:

```
/* 16-bit register (big endian format) */
  typedef union {
     word w;
     struct {
       byte high,low;
     } b;
  } TWREG;
```

Version Specific Information for 56800/E
For information on SDK types definitions please follow to the page SDK types.

Macros

```
  __DI()              - Disable global interrupts
  __EI()              - Enable global interrupts



  EnterCritical()     - It saves CCR register and disable
                        global interrupts
  ExitCritical()      - It restores CCR register saved
                        in EnterCritical()
```

For the list of macros available for Peripheral registers access please see the chapter 3.7.2 Direct Access to Peripheral Registers*3.7.2 Direct Access to Peripheral Registers* of the Processor Expert main help.

Constants

Methods Error Codes

The error codes are defined in PE_Error module. Error code value is 8bit unsigned byte. Range 0 - 127 is reserved for PE, 128 - 255 for user

ERR_OK	0	OK
ERR_SPEED	1	This device does not work in the active speed mode
ERR_RANGE	2	Parameter out of range

ERR_VALUE	3	Parameter of incorrect value
ERR_OVERFLOW	4	Timer overflow
ERR_MATH	5	Overflow during evaluation
ERR_ENABLED	6	Device is enabled
ERR_DISABLED	7	Device is disabled
ERR_BUSY	8	Device is busy
ERR_NOTAVAIL	9	Requested value not available
ERR_RXEMPTY	10	No data in receiver
ERR_TXFULL	11	Transmitter is full
ERR_BUSOFF	12	Bus not available
ERR_OVERRUN	13	Overrun is present
ERR_FRAMING	14	Framing error is detected
ERR_PARITY	15	Parity error is detected
ERR_NOISE	16	Noise error is detected
ERR_IDLE	17	Idle error is detected
ERR_FAULT	18	Fault error is detected
ERR_BREAK	19	Break char is received during communication
ERR_CRC	20	CRC error is detected
ERR_ARBITR	21	A node loses arbitration. This error occurs if two nodes start transmission at the same time
ERR_PROTECT	22	Protection error is detected.
ERR_UNDERFLOW	23	Underflow error is detected.
ERR_UNDERRUN	24	Underrun error is detected.
ERR_COMMON	25	General unspecified error of a device. The user can get a specific error code using the method GetError.

Version Specific Information for 56800/E
For information on SDK constants definitions please follow to the page SDK types.

3.5.2.1. 56800/E Additional Types For SDK Beans

The following types definitions are generated into the file *PETypes.h* in the Processor Expert for 56800/E. These types are intended to be used with the algorithms coming from the original SDK library. For more details please refer to the appropriate beans documentation.

```
typedef union
{
        struct
        {
          UWord16 LSBpart;
          Word16 MSBpart;
        } RegParts;

        Word32 Reg32bit;

} decoder_uReg32bit;

typedef struct
{
        union { Word16 PositionDifferenceHoldReg;
                Word16 posdh; };
        union { Word16 RevolutionHoldReg;
                Word16 revh; };
        union { decoder_uReg32bit PositionHoldReg;
                Word32 posh; };
}decoder_sState;

typedef struct
{
        UWord16 EncPulses;
        UWord16 RevolutionScale;

        Int16   scaleDiffPosCoef;
        UInt16  scalePosCoef;
        Int16   normDiffPosCoef;
        Int16   normPosCoef;
}decoder_sEncScale;

typedef struct
{
        UWord16 Index    :1;
        UWord16 PhaseB   :1;
        UWord16 PhaseA   :1;
        UWord16 Reserved :13;
}decoder_sEncSignals;

typedef union{
        decoder_sEncSignals  EncSignals;
        UWord16 Value;
} decoder_uEncSignals;


/******************************************************************************
*
* This Motor Control section contains generally useful and generic
* types that are used throughout the domain of motor control.
*

******************************************************************************/
/* Fractional data types for portability */
typedef short          Frac16;
typedef long           Frac32;

typedef enum
{
        mcPhaseA,
        mcPhaseB,
        mcPhaseC
} mc_ePhaseType;

typedef struct
{
        Frac16 PhaseA;
        Frac16 PhaseB;
        Frac16 PhaseC;
} mc_s3PhaseSystem;

/* general types, primary used in FOC */

typedef struct
{
        Frac16 alpha;
        Frac16 beta;
} mc_sPhase;

typedef struct
{
        Frac16 sine;
        Frac16 cosine;
} mc_sAngle;

typedef struct
{
        Frac16 d_axis;
        Frac16 q_axis;
} mc_sDQsystem;
```

```
typedef struct
{
        Frac16 psi_Rd;
        Frac16 omega_field;
        Frac16 i_Sd;
        Frac16 i_Sq;
} mc_sDQEstabl;

typedef UWord16 mc_tPWMSignalMask;
/*  pwm_tSignalMask contains six control bits
    representing six PWM signals, shown below.
    The bits can be combined in a numerical value
    that represents the union of the appropriate
    bits.  For example, the value 0x15 indicates
    that PWM signals 0, 2, and 4 are set.
*/

/* general types, primary used in PI, PID and other controllers */

typedef struct
{
   Word16 ProportionalGain;
   Word16 ProportionalGainScale;
   Word16 IntegralGain;
   Word16 IntegralGainScale;
   Word16 DerivativeGain;
   Word16 DerivativeGainScale;
   Word16 PositivePIDLimit;
   Word16 NegativePIDLimit;
   Word16 IntegralPortionK_1;
   Word16 InputErrorK_1;
}mc_sPIDparams;

typedef struct
{
   Word16 ProportionalGain;
   Word16 ProportionalGainScale;
   Word16 IntegralGain;
   Word16 IntegralGainScale;
   Word16 PositivePILimit;
   Word16 NegativePILimit;
   Word16 IntegralPortionK_1;
}mc_sPIparams;

#endif /* __PE_Types_H */

#define MC_PWM_SIGNAL_0       0x0001
#define MC_PWM_SIGNAL_1       0x0002
#define MC_PWM_SIGNAL_2       0x0004
```

```
#define MC_PWM_SIGNAL_3       0x0008
#define MC_PWM_SIGNAL_4       0x0010
#define MC_PWM_SIGNAL_5       0x0020
#define MC_PWM_NO_SIGNALS     0x0000     /* No (none) PWM signals */
#define MC_PWM_ALL_SIGNALS   (MC_PWM_SIGNAL_0 | \
                              MC_PWM_SIGNAL_1 | \
                              MC_PWM_SIGNAL_2 | \
                              MC_PWM_SIGNAL_3 | \
                              MC_PWM_SIGNAL_4 | \
                              MC_PWM_SIGNAL_5)
```

3.5.3. Typical Usage of the Bean in the User Code

This chapter describes usage of methods and events that are defined in most hardware beans. Usage of other bean specific methods is described in the bean documentation, in the section "Typical Usage" *(if supported)*.

In the following examples please assume a bean named "B1".

Peripheral Initialization Beans

Peripheral Initialization Beans are the beans of the lowest level of peripheral abstraction. These beans contain only one method Init providing the initialization of the used peripheral. See chapter *3.5.3.1 Typical Usage of Peripheral Initialization Beans* for details.

Methods Enable, Disable

Most of the hardware beans support the methods Enable and Disable. These methods enable or disable peripheral functionality, which causes disabling of functionality of the bean as well.
Hint: Disabling of the peripheral functionality may save CPU resources.

Overview of the method behavior according to the bean type:

- Timer beans: timer counter is stopped if it is not shared with another bean. If the timer is shared, the interrupt may be disabled (if it is not also shared).
- Communication beans (like serial or CAN communication): peripheral is disabled.
- Conversion beans (A/D, D/A): converter is disabled. The conversion is restarted by Enable.

If the bean is disabled, some methods may not be used. Please refer to beans documentation for details.

MAIN.C

```
  void main(void)
  {
    ...
    B1_Enable();   /* enable the bean functionality */
                /* handle the bean data or settings */
    B1_Disable(); /* disable the bean functionality */
    ...
  }
```

Methods EnableEvent, DisableEvent

These methods enable or disable invocation of all bean events. These methods are usually supported only if the bean services any interrupt vector.

The method DisableEvent may cause disabling of the interrupt, if it is not required by the bean functionality or shared with another bean. The method usually does not disable either peripheral or the bean functionality.

MAIN.C

```
void main(void)
{
  ...
  B1_EnableEvent();  /* enable the bean events */
                     /* bean events may be invoked */
  B1_DisableEvent(); /* disable the bean events */
                     /* bean events are disabled */
  ...
}
```

Events BeforeNewSpeed, AfterNewSpeed

Timed beans which depends on the CPU clock (such as timer, communication and conversion beans), may support speed modes defined in the CPU bean (in EXPERT view level). The event BeforeNewSpeed is invoked before the speed mode change and AfterNewSpeed is invoked after the speed mode change. Speed mode may be changed using CPU bean methods SetHigh, SetLow or SetSlow.

EVENT.C

```
int changing_speed_mode = 0;

void B1_BeforeNewSpeed(void)
{
  ++changing_speed_mode;
}

void B1_AfterNewSpeed(void)
{
  --changing_speed_mode;
}
```

Note: If the speed mode is not supported by the bean, the bean functionality is disabled (as if the method Disable is used). If the supported speed mode is selected again, the bean status is restored.

3.5.3.1. Typical Usage of Peripheral Initialization Beans

Init method

Init method that is defined in all Peripheral Initialization Beans. Init method contains a complete initialization of the peripheral according to the bean's settings.

In the following examples, let's assume a bean named "Init1" has been added into the project.

The Init method of the Peripheral Initialization bean can be used in two ways.

- The Init method is called by Processor Expert
- The Init method is called by the user in his/her module

Automatic calling of Init
The user can let Processor Expert call the Init method automatically by selecting "yes" for the **Call Init method** in the Initialization group of the Bean's properties.

When this option is set, Processor Expert places the call of the Init method into the *PE_low_level_init* function of the CPU.c module.

Manual calling of Init
Add the call of the Init method into the user's code (for example in main module).

Enter the following line into the main module file:

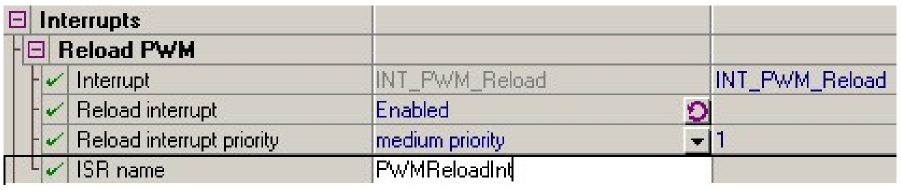
```
Init1_Init();
```

Put the Init method right below the PE_low_level_init call.

```
void main(void)
{
  /*** Processor Expert internal initialization. ***/
  PE_low_level_init();
  /*** End of Processor Expert internal initialization. ***/
    Init1_Init();
  for(;;) {}
}
```

Interrupt Handling

Some Peripheral Initialization beans allow the initialization of an interrupt service routine. Interrupt(s) can be enabled in initialization code using appropriate properties that can be usually found within a group *Interrupts*.
After enabling, the specification of an Interrupt Service Routine (ISR) name using the *ISR name* property is required. This name is generated to Interrupt Vector table during the code generation process. See chapter *3.3.1.1 Interrupt Vector Table* for details.
Please notice that if the ISR name is filled it is generated into the Interrupt Vector Table even if the interrupt property is disabled.



Interrupts		
Reload PWM		
✓ Interrupt	INT_PWM_Reload	INT_PWM_Reload
✓ Reload interrupt	Enabled	
✓ Reload interrupt priority	medium priority	1
✓ ISR name	PWMReloadInt	

Figure 3.19 - Example Of The Interrupt Setup

Enabling/disabling peripheral interrupts during runtime has to be done by user's code (for example by utilizing PESL or direct register access macros) because The *Peripheral Initialization Beans* do not offer any methods for interrupt handling.

The ISR with the specified name has to be declared according to the compiler conventions and fully implemented by the user. Declarations of the ISRs that already do not exist can be generated automatically by PE during the code generation process into the Event module of the CPU bean (name of the CPU event module could be changed in the Advanced view mode of the Bean Inspector) if the project option **Project Options | Generate ISR** is enabled (this option is visible in the Advanced level of view).

Notice for 56800/E version users: ISRs generated by Processor Expert contain the fast interrupt handling instructions if the interrupt priority is specified as fast interrupt.

3.5.4. User Changes in Generated Code

It's necessary to say at the beginning of the chapter, that modification of the generated code may be done only at user's own risk. Generated code was thoroughly tested by the skilled developers and the functionality of the modified code cannot be guaranteed. We strongly don't recommend modification of the generated code to the beginners. See more information for generated modules in chapter Code Generation.

To support user changes in the bean modules, Processor Expert supports the following features:

1. CH file - manifest constants

Processor Expert automatically produce list of manifest constants for all beans, which encapsulate any CPU peripheral and modifies any of the CPU control registers. The name of each manifest constant is in the following format:

C_*[bean+method]*_reg_*[register name][additional-info]*

where

- *[bean+method]* is name of bean (and optionally also bean's method),
- *[register name]* is name of the control register,
- *[additional-info]* is additional information about usage of the value (modification of one bit, bits mask or whole value).

These constants may be used to write user code, which reflects the bean settings. Once the constant is generated into the CH file, it is preserved there even it is already not used in the bean module. The most important advantage of these constants is, that small changes in the bean settings (for example timing) does not cause change of the bean module, but the only CH file is changed.

The name of the CH file is derived from the CPU bean name ([CPUbean].CH). To generate CH file it is necessary to set the following option: **Project Options | Generate manifest constants**. CH file is generated also while smart generation of bean modules code, see paragraph 3 for details. CH file is always overwritten during the code generation.

2. Mode of code generation for bean modules

It's possible to select mode of the code generation for each bean, the following options can be found in the bean's pop-up menu in the **project panel**:

Code Generation

- **Always Write Generated Bean Modules** (default) - generated bean modules are always written to disk and any existing previous module is overwritten
- **Preserve User Changed in Generated Bean Modules** - smart detection of user changes. See paragraph 4 for details.
- **Don't Write Generated Bean Modules** - the code from bean is not generated. Any initialization code of the bean, which resides in the CPU bean, interrupt vector table and shared modules are updated.

The mode of code generation is indicated as a bean's status in the project panel. This mode influences only the generation of bean modules (bean.c, bean.h).

3. Mode of code generation for non-bean modules

Processor Expert also allows to enable/disable generation of the modules that not related to a specific bean or that are common for several beans. This option can be configured in the pop-up menu of the module in the Project Panel. Current enable/disable state of this option is signalled by the icon near the bean module name in the Project Panel (- enabled, - disabled).

Code Generation

- **Always Write** (default) - generated bean modules are always written to disk and any existing previous module is overwritten
- **Don't Write** - the content of the module is not overwritten. *Please notice that this can lead to malfunction of beans dependent on the module when the module update is required due to bean's settings change.*

4. Smart generation of the code of bean modules

Notice: Smart user changes preservation is available only in the 56F800/E version.

To enable smart generation of the bean modules, it's necessary to set the project option **Project Options | Preserve user changes**. (option **Project Options | Generate manifest constants** is turned on automatically). After setting this option it's necessary to generate the code and after that Processor Expert can detect and preserve changes in the generated code of bean modules. In this mode it is still possible to select for each bean, if the code will be overwritten, not written or if the user changes will be preserved (see paragraph 2). In case of first use of the bean, the generated code will be always overwritten.
The user can make any change in the generated bean module. If the Processor Expert detects during code generation, that the bean module was changed, the user is informed immediately. Status icon of the bean with changed module is red . Until the generated code is not changed, the user changes are completely preserved.

The generated code of the bean module may be changed from several reasons:

- the user changed the bean settings

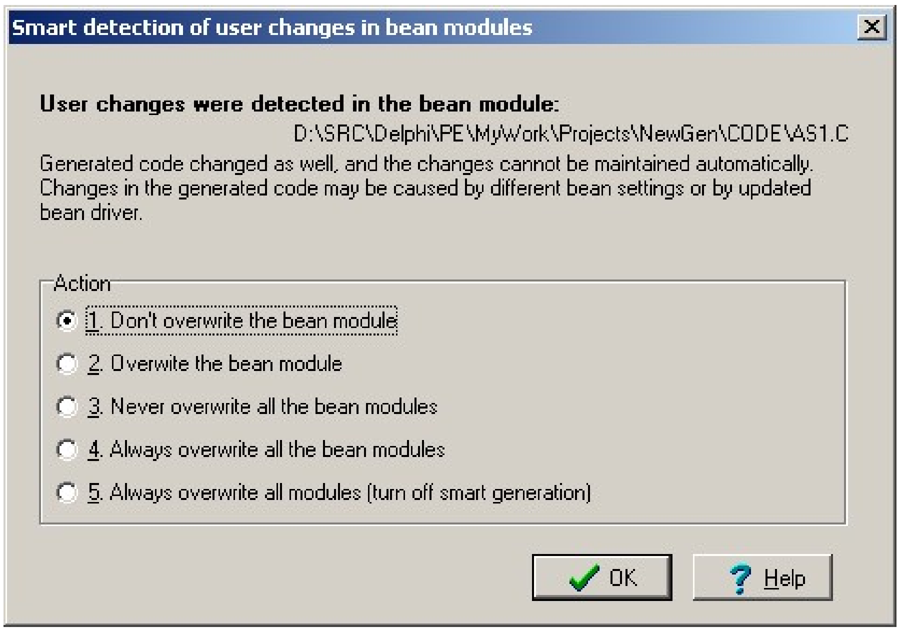
- the user changed settings of another bean, and the change is reflected to the bean's module
- the bean driver was changed (updated)

The following changes in the generated code may be updated automatically into the bean's module:

- new method is generated into the bean's module - this method is automatically added into the bean's module
- any method from the bean's module is not generated - if the method code is not changed in the user code, the method is automatically removed from the bean's module.
- If the code of the generated method has changed and it wasn't changed by the user, the code of method is automatically updated.

In all other cases the user must select how to handle the changed code (see the following picture):

- Don't overwrite the bean module (default) - the user changes are preserved and the generated code is ignored
- Overwrite the bean module - the user changes are discarded and the user module is rewritten by the new generated code
- Never overwrite all the bean modules - the mode of the code generation for the bean is "Don't Write Generated Code", see paragraph 2 for details
- Always overwrite all the bean modules - the mode of the code generation for the bean is "Always Write Generated Code", see paragraph 2 for details
- Always overwrite all modules (turn off smart generation) - smart generation of the code if switched off, all user modules are overwritten.


Smart detection of user changes in bean modules

User changes were detected in the bean module:
D:\SRC\Delphi\PE\MyWork\Projects\NewGen\CODE\AS1.C
Generated code changed as well, and the changes cannot be maintained automatically. Changes in the generated code may be caused by different bean settings or by updated bean driver.

Action
[✓ 1. Don't overwrite the bean module]
2. Overwite the bean module
3. Never overwrite all the bean modules
4. Always overwrite all the bean modules
5. Always overwrite all modules (turn off smart generation)

Figure 3.20 - User Changes Handling Options

Viewing User Changes in a Bean Module

The user changes done in a bean module or bean header module can be viewed using a bean pop-up menu commands **Compare With Previously Generated Module** and **Compare With Previously Generated Header Module**. See chapter *2.3.4 Beans Pop-up Menus* for details. The user can also enable reviewing all changes done into the generated code after each code generation. See chapter *3.5.1.2 Tracking Changes in Generated Code* for details.

3.6. Embedded Bean Optimizations

This chapter describes how the size and speed of the code could be optimized by choosing right bean for the specific task. It should also give an advice how to setup beans to produce optimized code. The optimizations that will be described regard only the High or Low level beans, not the Peripheral Initialization beans.

Please follow to sub-chapters for more details on

- General Optimizations
- General Port I/O Optimizations
- Timer Beans Optimizations
- Code Size Optimization of Communication Beans

3.6.1. General Optimizations

This chapter contains general hints and instructions how to setup Processor Expert and beans to generate optimized code. The following optimization regards only the High or Low level beans, not the Peripheral Initialization beans.

Disabling unused methods

Notice: These optimization are not usable for the Peripheral Initialization Beans
When Processor Expert generates the code certain methods and events are enabled by default setting, even when the methods or events are not needed in the application, and thus while they are unused, its code still can take memory. Basically, the unused methods code is dead stripped by the linker but when the dependency among methods is complex some code should not be dead stripped. When useless methods or events are enabled the generated code can contain spare source code because of these unused methods or events. Moreover some methods can be replaced by more efficient methods that are for special purposes and therefore these methods are not enabled by default.

Disabling unused beans

Disable unused and test purpose beans or remove them from the project. Disabling of these beans is sufficient because the useless code is removed but the bean setting remains in the project. If these beans are required for later testing then add a new configuration to the project and disable these useless beans in the new configuration only (the previous configuration will be used when the application is tested again). Moreover if it is required to use the same bean with different setting in several configurations its possible to add one bean for each configuration with same name and different setting.

Speed modes

Notice: These optimizations are not usable for the Peripheral Initialization Beans
Timed beans which depends on the CPU clock (such as timer, communication and conversion beans), may support speed modes defined in the CPU bean (in EXPERT view level). The Processor Expert allows the user to set closest values for the bean timing in all speed modes (if possible). If the requested timing is not supported by the bean, for example if the CPU clock is too low for the correct function of the bean, the bean can be disabled for the appropriate speed mode. The mode can be switched in the runtime by a CPU method. The bean timing is then automatically configured for the appropriate speed mode or the bean is disabled (according to the setting). Note, however, that use of speed modes adds extra code to the application. This code must be included to support different clock rates. See speed mode details here.

See Configuration Inspector for more optimization settings.

See chapter Embedded Beans Optimizations for details on choosing and setting the beans to achieve optimized code.

3.6.2. General Port I/O Optimizations

Notice: These optimizations are not usable for the Peripheral Initialization Beans

ByteIO Bean Versus BitsIO Bean

ByteIO bean instead of BitsIO bean should be used when whole port is accessed. The BitsIO bean is intended for accessing only part of the port (e.g. 4 bits of 8- bit port)

Using the BitsIO bean results more complex code because this bean provides more general code for the methods, which allows access to only some of the bits of the port. On the other side, the ByteIO bean provides access only to the whole port and thus the resulted code is optimized for such type of access.

BitsIO bean versus BitIO beans

In the case of using only a part of the port the multiple BitIO beans could be used. A better solution is to use the BitsIO bean replacing multiple calls of BitIO bean's methods. The application code consist only of one method call and is smaller and faster.

3.6.3. Timer Beans Optimizations

Notice: These optimizations are not usable for the Peripheral Initialization Beans
For better code size performance it's recommended do not use a bigger counter/reload/compare register for timer than is necessary. Otherwise the code size generated by a bean may be increased (e.g. For 8-bit timer choose 8bit timer register).

In some cases several timing periods are required when using timers (For example the TimerInt bean). The Processor Expert allows changing the timer period during run-time using several ways (note that this is an advanced option and the Bean Inspector Items visibility must be set to at least 'ADVANCED').

These ways of changing the run-time period of timer requires various amount of code and thus the total application code size is influenced by the method chosen. **When the period must be changed during run-time**, use fixed values for period instead of an interval if possible to save code. There are two possibilities (See chapter *2.5.3.1 Dialog Box for Timing Settings* for details.):

- **From list of values** - this allow to specify several (but fixed in run-time) number for given periods. This allows only exact values - modes, listed in the listbox. The resulted code for changing the period is less complex than using an interval.
- **From time interval** - this is an alternative to using 'list of values', which requires more code. Using an interval allows setting whatever value specified by the bean during run-time. This code re-calculates the time period to the CPU ticks and this value is used when changing the timer period.

If the application requires only a few different timing periods, even if the functionality is the same for both cases, the correct usage of list of periods produces smaller code compared to code using an interval.

3.6.4. Code Size Optimization of Communication Beans

Notice: These optimizations are not usable for the Peripheral Initialization Beans
Communication beans should be used with the smallest possible buffer. Thus the user should compute or check the maximum size of the buffer during execution of the application. For this purpose the method GetCharsInTxBuffer/GetCharsInTxBuffer (AsynchroSerial bean), which gets current size of a used buffer, can be used after each calling of the SendBlock/RecvBlock method.

Use **interrupts** if you require faster application response. The interrupt routine is performed only at the event time, i.e. the code does not check if a character is sent or received. Thus the saved CPU time can be used by another process and application is faster.

Use **polling mode** instead interrupts if you require less code because usually overhead of interrupts is bigger than overhead of methods in polling mode. But the polling mode is not suitable for all cases. For example when you use the SCI communication for sending of the data only and a character is sent once in a while then it is better to use the polling mode instead of using interrupt because it saves the code size, i.e. when the interrupt is used an interrupt subroutine is needed and code size is increased.

Examples:

A module of an application sends once in a while one character to another device through the SCI channel. If the delay between two characters is sufficient to sent one character at a time then the polling mode of the SCI (the AsynchroSerial bean) should be used in this case.

A module of an application communicates with another device, i.e. it sends several characters at one time and receives characters from the device. Thus the interrupt mode of the SCI (the AsynchroSerial bean) should be used in this case because when a character is received the interrupt is invoked and the underlying process of the application need not check if a character is received. When a buffer for sending is used, the characters are saved into the buffer and AsynchroSerial's service routine of the interrupt sends these characters without additional code of the application.

Note: The polling mode of the bean is switched on by disabling of the Interrupt service of the bean (AsynchroSerial, AsynchroMaster, AsynchroSlave, …)

3.7. Low-level Access to Peripherals

In some cases, a non-standard use of the peripheral could be required and it might be more efficient to write a custom peripheral driver from scratch than to use the bean. In addition, there are special features present only on a particular chip derivative (not supported by the bean) that could make the user routines more effective; however, the portability of such code is reduced.

Peripheral Initialization

It is possible to use Processor Expert to generate only the initialization code (function) for a peripheral using a Peripheral initialization beans. The user can choose a suitable Peripheral initialization bean for the given peripheral using a Bean Selector under a Peripherals tab. See chapter *2.4 Bean Selector* for details. The initial values which will be set to the peripheral control registers could be viewed by the Peripheral Initialization window. See chapter *2.14 Peripheral Initialization* for details.

Peripheral Driver Implementation

The rest of the peripheral driver can be implemented by the user using one of the following approaches:

- Processor Expert System Library (PESL)
- Direct control of the CPU registers

Warning: Incorrect use of PESL or change in registers of the peripheral, which is controlled by any Bean driver can cause the incorrect Bean driver function.

3.7.1. Processor Expert System Library

Notice: PESL is available only in the 56F800/E version.

PESL (Processor Expert System Library) is dedicated to power programmers, who are familiar with CPU architecture - each bit and each register. PESL provides macros to access the peripherals directly. It should be used only in special cases when the low-level approach is necessary.

PESL is peripheral oriented and complements with Embedded Beans, which are functionality oriented. While Embedded Beans provide very high level of project portability by stable API and inheritance feature across different CPU/DSP/PPC architectures, PESL is more architecture dependent.

PESL commands grouped by the related peripheral can be found in Processor Expert Project Panel in PESL folder.

Convention for PESL macros

Each name of the PESL macro consists of the following parts:

```
PESL(device name, command, parameter)
```

Example:

```
PESL(SCI0, SCI_SET_BAUDRATE, 0);
```

Using PESL and Peripheral Initialization Beans

For every Peripheral Initialization Bean (for details see the chapter 3.2.1.1 Bean Categories) there is a C macro defined by Processor Expert with the name ***bean name*** **_DEVICE** . This macro results to the name of the peripheral selected in the bean named *'bean name'*. Using this macro instead of a real peripheral name allows a peripheral to be changed later by changing the bean property without modifying the PESL commands in user code.

Example:
Let's expect we have a bean Init_SCI named *SCIBEAN1*:

```
PESL(SCIBEAN1_DEVICE, SCI_SET_BAUDRATE, 1);
```

Processor Expert shows the list of the available PESL commands as a subtree of the Peripheral Initialization bean in the Project Panel (please see the chapter 2.3 Project Panel for details). User can drag and drop the commands into the code from this tree. The PESL commands created this way use the ***bean name*****_DEVICE** macro instead of a specific peripheral name.

PESL Commands Reference

For details on PESL, its commands and parameters, see PESL Library user manual using the **Help** command of PESL project panel folder pop-up menu.

3.7.2. Direct Access to Peripheral Registers

The direct control of the Peripheral's registers is a low-level way of creating peripheral driver which requires a good knowledge of the target platform and the code is typically not portable to different platform. However, in some cases is this method more effective or even necessary to use (in the case of special chip features not encapsulated within the Embedded bean implementation). See chapter *3.7 Low-level Access to Peripherals* for details.

The common basic peripheral operations are encapsulated by the PESL library commands which is effectively implemented using the simple control register writes. See chapter 3.7.1 Processor Expert System Library for details.

Register Access Macros

Processor Expert defines a set of C macros providing an effective access to a specified register or its part. The definitions of all these macros are in the **PE_Types.h** . The declaration of the registers which could be read/written by the macros is present in the file **IO_Map.h**.

Whole Register Access Macros

- **getReg{w}** (*RegName*) - Reads the register content
- **setReg{w}** (*RegName, RegValue*) - Sets the register content

Register Part Access Macros

- **testReg{w}Bits** (*RegName, GetMask*) - Tests the masked bits for non-zero value
- **clrReg{w}Bits** (*RegName, ClrMask*) - Sets a specified bits to 0.
- **setReg{w}Bits** (*RegName, SetMask*) - Sets a specified bits to 1.
- **invertReg{w}Bits** (*RegName, InvMask*) - Inverts a specified bits.

- **clrSetReg{w}Bits** (*RegName, ClrMask, SetMask*) - Clears bits specified by ClrMask and sets bits specified by SetMask

Access To Named Bits

- **testReg{w}Bit** (*RegName, BitName*) - Tests whether the bit is set.
- **setReg{w}Bit** (*RegName, BitName*) - Sets the bit to 1.
- **clrReg{w}Bit** (*RegName, BitName*) - Sets the bit to 0.
- **invertReg{w}Bit** (*RegName, BitName*) - Inverts the bit.

Access To Named Groups of Bits

- **testReg{w}BitGroup** (*RegName, GroupName*) - Test a group of the bit for non-zero value
- **getReg{w}BitGroupVal**(*RegName, GroupName*) - Read a value of the bits in group
- **setReg{w}BitGroupVal** (*RegName, GroupName, GroupVal*) - Sets the group of the bits to the specified value.

RegName - Register name
BitName - Name of the bit
GroupName - Name of the group
BitMask - Mask of the bit
BitsMask - Mask specifying one or more bits
BitsVal - Value of the bits masked by BitsMask
GroupMask - Mask of the group of bits
GetMask - Mask for reading bit(s)
ClrMask - Mask for clearing bit(s)
SetMask - Mask for setting bit(s)
InvMask - Mask for inverting bit(s)
RegValue - Value of the whole register
BitValue - Value of the bit (0 for 0, anything else = 1)
{w} - Width of the register (8, 16, 32). The available width of the registers depends on used platform.

Example

Assume that we have a CPU which has a PWMA channel and it is required to set three bits (0,1,5) in the PWMA_PMCTL to 1. We use the following line:

```
setRegBits(PWMA_PMCTL,35);           /* Run counter */
```

4. Processor Expert Tutorials

This tutorial is provided for embedded system designers who wish to learn quickly how to use the exclusive features of Processor Expert. Reading this tutorial may be all you need to start using Processor Expert for your own application.

The following tutorials are available:

HC08 Project 1
HC08 Project 2

4.1. Tutorial Project 1 for Freescale HC08 Microcontrollers

This simple animated tutorial describes a periodically blinking LED project. The LED is connected to one pin of the CPU and it is controlled by a periodical timer interrupt. Please follow the instructions and animations showing the actions.

Click here to start the tutorial

4.2. Tutorial Project 2 for Freescale HC08 Microcontrollers

This tutorial describes a demo project of a simple LED controller. The LED controller has two color LEDs - a red and a green one - and one command button.

How it works

The button sends commands (external interrupts) to CPU through one pin and the CPU switches the red and green LEDs lights on or off. If you press the key you can see that lights of two LEDs have been changed. One of them is switched off and the other one is switched on. At the beginning the green LED is on and the red one is off.

Minimal required hardware design

In the demo application the following components are used:

1. CPU MC68HC908AZ60 Freescale HC08 processor family
2. Red LED - connected to CPU output pin PTB0
3. Green LED - connected to CPU output pin PTB1
4. Button - connected to CPU input pin PTG0

Beans

This simple demo-project uses the following beans:

1. MC68HC908AZ60 - CPU bean (Freescale HC08 processor family)
2. PinIO - General 1-bit input/output bean - outputs to LEDs. The LEDs receive 1 bit data which specifies whether the light should switch on or off (value 0 = switch off, value 1 = switch on).
3. ExtInterrupt - External Interrupt bean - interrupt from button. Pressing the button calls an external interrupt which switches the state of the LEDs (on/off).

steps

There are step-by-step instructions how to create this tutorial project. This tutorial goes through the following steps:

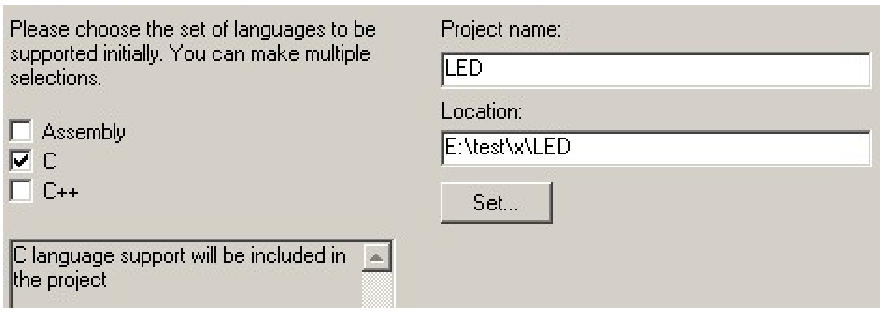
1. Creating a new project
2. Adding beans
3. Code Generation
4. Adding the On-Event Code

Note: This demo project does not care about non-defined states on the output of the key during the process of key pressing. This may result in the fact that state of two LEDs stays apparently unchanged.

4.2.1. Tutorial for Freescale HC08 Project 2 Step 1

Creating a New Project

1. If you have the CodeWarrior already running, click on the command **New Project...** in the menu *File* in the CodeWarrior window in order to create a new project. Otherwise, start the CodeWarrior and click on the button **Create empty project**.
2. The Procect Wizard dialog window appears. Enter the name of the project - "LED". Click on the *next* button.



Please choose the set of languages to be supported initially. You can make multiple selections.

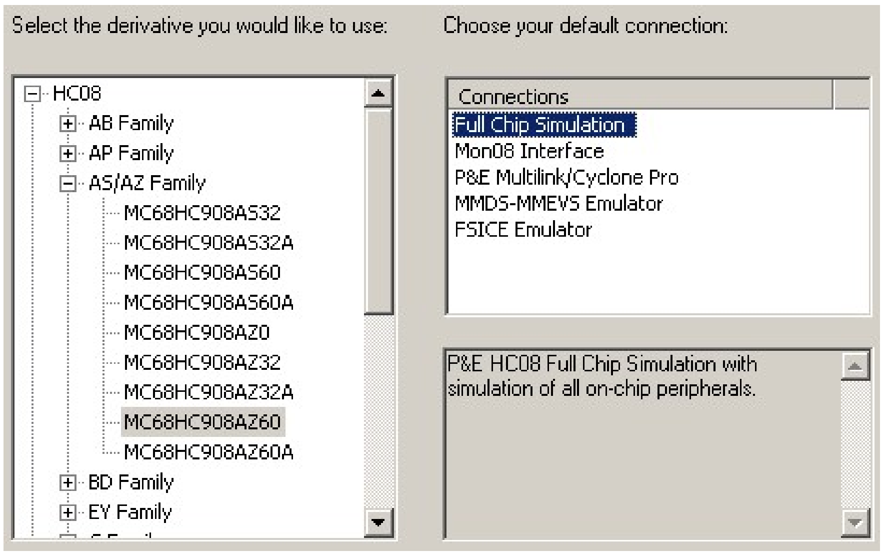
- [ ] Assembly
- [✓] C
- [ ] C++

C language support will be included in the project

Project name: LED

Location: E:\test\x\LED

3. Select *MC68HC908AZ60* CPU and the *Full Chip Simulation* as the default connection. Click on the *next* button.



Select the derivative you would like to use:

```
HC08
  AB Family
  AP Family
  AS/AZ Family
    MC68HC908AS32
    MC68HC908AS32A
    MC68HC908AS60
    MC68HC908AS60A
    MC68HC908AZ0
    MC68HC908AZ32
    MC68HC908AZ32A
    MC68HC908AZ60
    MC68HC908AZ60A
  BD Family
  EY Family
```

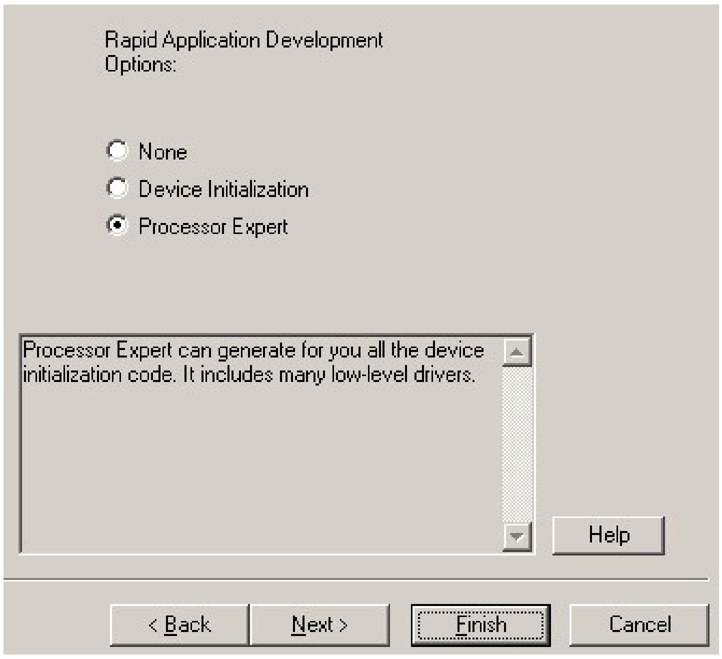
Choose your default connection:

Connections
- Full Chip Simulation
- Mon08 Interface
- P&E Multilink/Cyclone Pro
- MMDS-MMEVS Emulator
- FSICE Emulator

P&E HC08 Full Chip Simulation with simulation of all on-chip peripherals.

4. Skip addition of existing files by clicking on the "next" button.

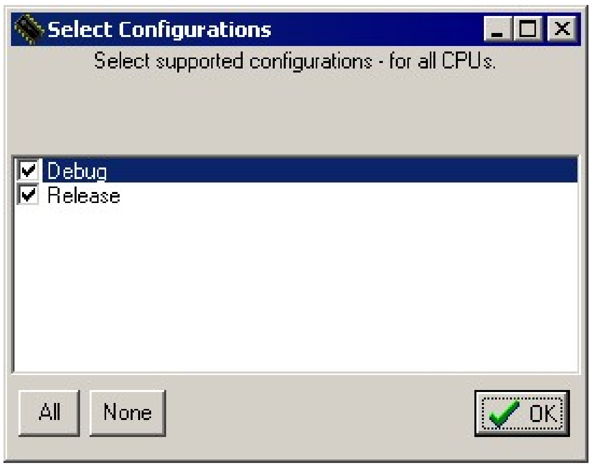
5. Select the *Processor Expert* from *Rapid Application Development Options* and click on the *finish* button.


Rapid Application Development Options:

( ) None
( ) Device Initialization
(●) Processor Expert

Processor Expert can generate for you all the device initialization code. It includes many low-level drivers.

6. Confirm the dialog *Select configurations* by clicking on the "OK" button.


Select Configurations

Select supported configurations - for all CPUs.

☑ Debug
☑ Release

Now the new empty project is created and ready for adding new beans.

Next step

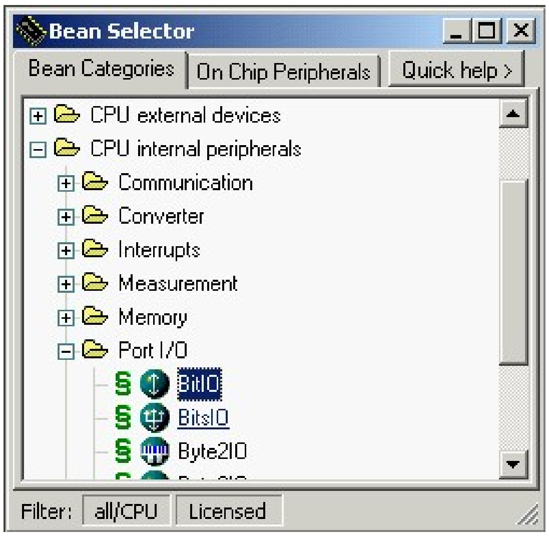
Go to Step 2 - Adding beans to the project.

4.2.2. Tutorial for Freescale HC08 Project 2 Step 2

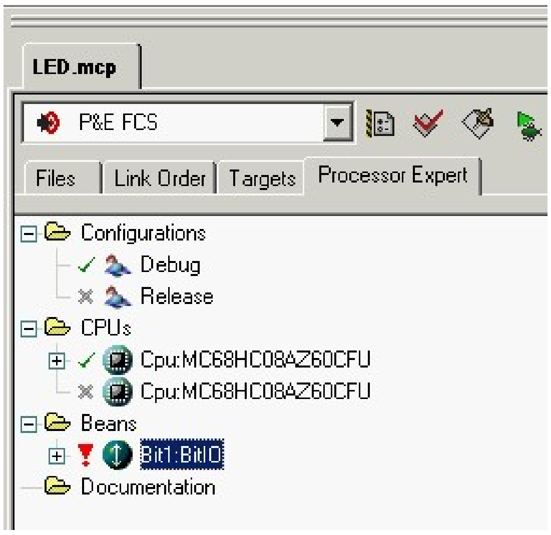
Adding Beans to the Project

In the context of the LED controller, you will add two **BitIO** beans for the Red LED and the Green LED, and an **ExtInt** bean for the Button.

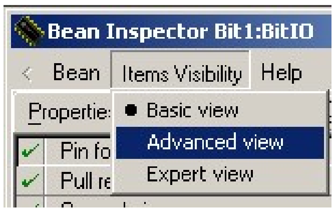
1. If the Bean Selector window is not already opened, open it using menu **Processor Expert | View | Bean Selector**
2. Double click the **BitIO** bean in the folder PortI/O (subfolder of CPU Internal Peripherals folder) in the Bean selector window.



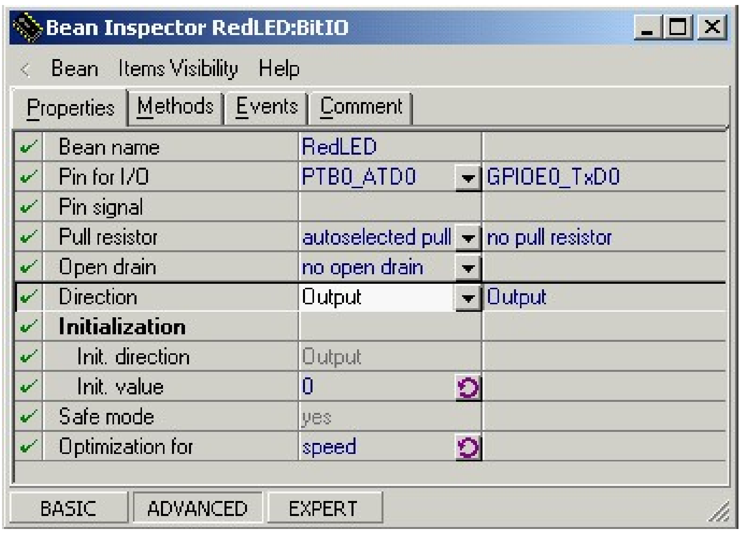
3. You will be asked whether to enable the bean in all configurations. Choose Yes.
4. New bean is added to the project. Switch to Processor Expert project tab in codewarrior's project panel. (See picture below). *Don't worry about red exclamation mark beside a new bean. It means that error is present - bean has not been set-up properly yet.*



5. Open the **Bean Inspector** window by double click on the new bean (Bit1:BitIO) in Processor Expert project panel (if the Bean Inspector hasn't been opened automatically). *Automatical opening of the bean inspector is influenced by environment settings (See chapter 2.1.1 Processor Expert Options* for details.)
6. Select **Items visibility | Advanced view** in the pop-up menu of the Bean Inspector window in order to display detailed settings of the bean. It is necessary for the following steps. See picture below.

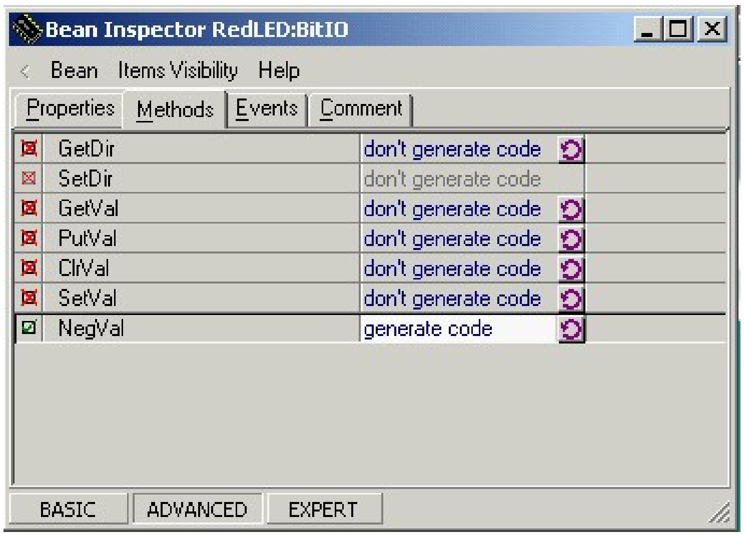


7. Using Bean inspector set the bean properies as follows:
   - **Bean name:** type **RedLED** into the edit box.
   - **Pin for IO:** select **PTB0_ATD0**
   - **Direction:** select **Output**



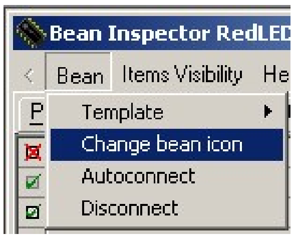
✓	Bean name	RedLED	
✓	Pin for I/O	PTB0_ATD0	GPIOE0_TxD0
✓	Pin signal		
✓	Pull resistor	autoselected pull	no pull resistor
✓	Open drain	no open drain	
✓	Direction	Output	Output
✓	**Initialization**		
✓	Init. direction	Output	
✓	Init. value	0	
✓	Safe mode	yes	
✓	Optimization for	speed	

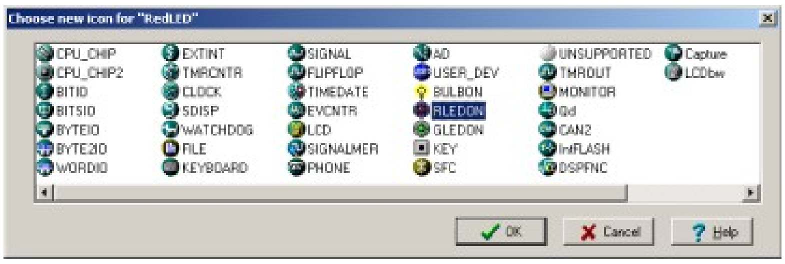
8. To setup generation of **methods** click on the Methods TAB and set all methods to "don't generate" and NegVal method to 'generate code'. See the following picture:



☒	GetDir	don't generate code
☒	SetDir	don't generate code
☒	GetVal	don't generate code
☒	PutVal	don't generate code
☒	ClrVal	don't generate code
☒	SetVal	don't generate code
✓	NegVal	generate code

9. click the **Change bean icon** item of the *Bean* menu in order to choose a new icon in the list. Select the **RLEDON** icon and click the **OK** button. See the following pictures:

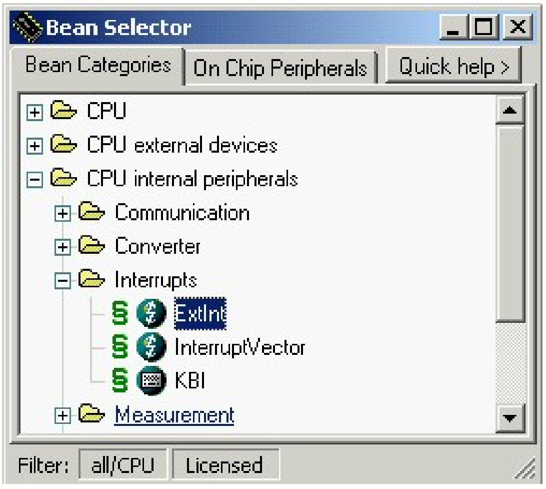




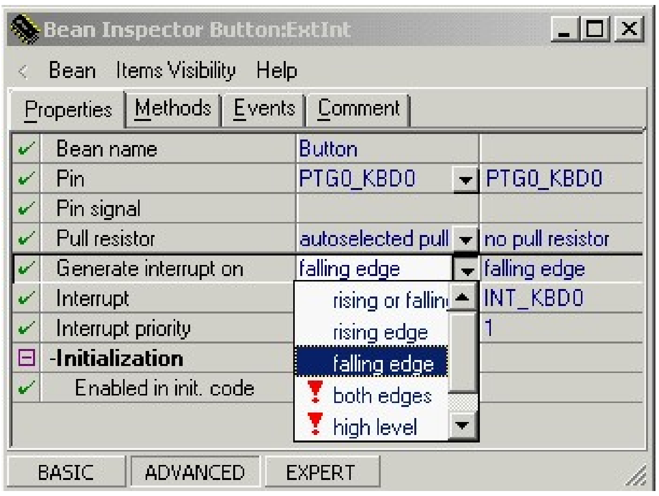
10. Using the procedure previously described for the red LED (steps a,b,c,d) add the **green LED** bean to the project. The difference from the redLED bean is in the bean's propeties (pin, name and initial value).
Set the bean properties as follows:
    - **Bean name:** type **GreenLED** into the edit box.
    - **Pin for IO:** select **PTB1_ATD1**
    - **Direction:** select **Output**
    - **Init. value:** select **1**

11. Click on the Methods TAB and switch all methods to "don't generate" and NegVal method to "generate code" using button on the right.
12. Click the **Change bean icon** item of the **Bean** menu in order to choose a new icon. Select the **GLEDON** icon and click the **OK** button.
13. Now is time to add bean handling **the button**. Open the **Bean Selector window** again and double click the

ExtInt icon on the folder Interrupts in the in order to add the bean to the project. See picture below:



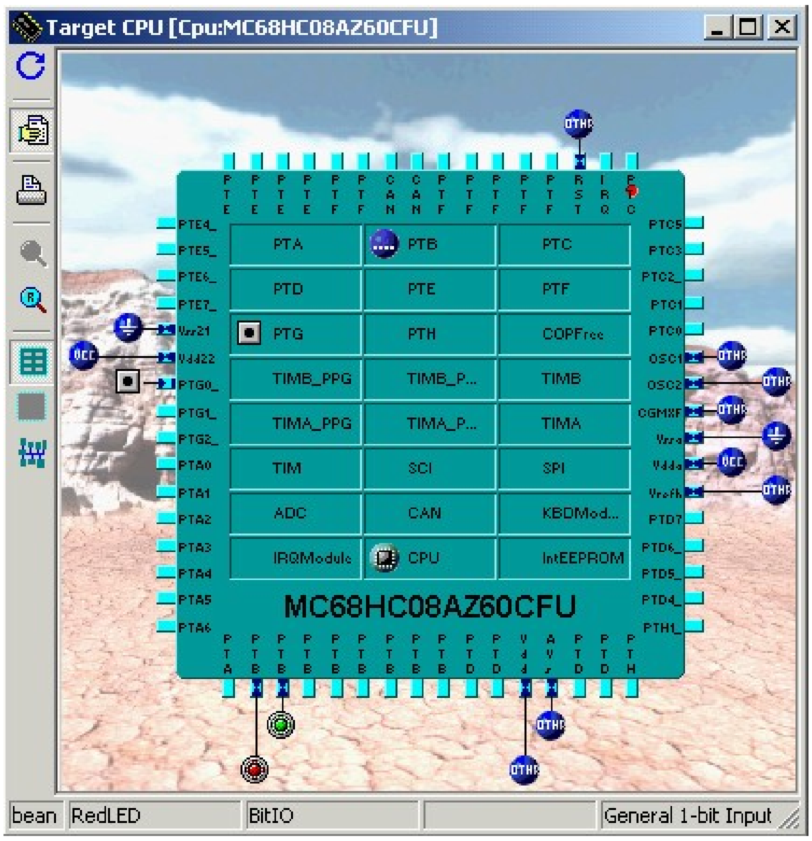
14. Open the **bean inspector** for the new bean (double click it in the project panel) and set the bean properties as follows:
    - **Bean name:** type **Button** into the edit box.
    - **Pin:** select **PTG0_KBD0**
    - **Generate interrupt on:** click the option in order to display the options. Select the **falling edge**.



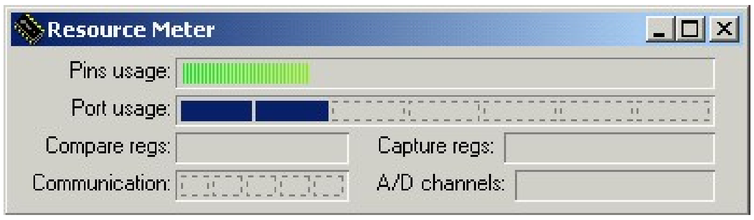
✓	Bean name	Button	
✓	Pin	PTG0_KBD0	PTG0_KBD0
✓	Pin signal		
✓	Pull resistor	autoselected pull	no pull resistor
✓	Generate interrupt on	falling edge	falling edge
✓	Interrupt		INT_KBD0
✓	Interrupt priority		1
⊟	-**Initialization**		
✓	Enabled in init. code		

15. Click on the Methods TAB and set **all methods to "don't generate"**.
16. Click the *Change bean icon* item of the *Bean* menu in order to choose a new icon in the list. Select the **KEY** icon and click the **OK** button.

You can see in the Target CPU window which pins of the chip are handled by the beans. You can easily identify LED beans by their specific icons. If the Target CPU window is not opened, use menu command ***Processor Expert | View | Target CPU package***



*After adding all beans, click **Processor Expert | View | Resource Meter** in order to open the Resource Meter window and see remaining available resources of the chip.*



Next step

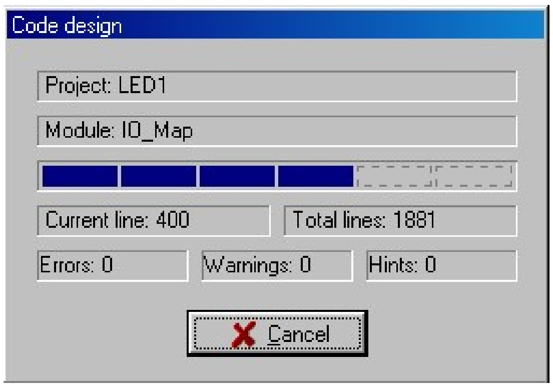
Go to Step 3 - Code Generation.

4.2.3. Tutorial for Freescale HC08 Project 2 Step 3

Code Generation

1. Click on the command **Processor expert | Code Generation 'Led.mcp'** in the CodeWarrior main menu in order to run the code generation process
   The code generation window shows the current state of code generation.

   Note: *There shouldn't be any errors in the Error window before code generation.*



Code design

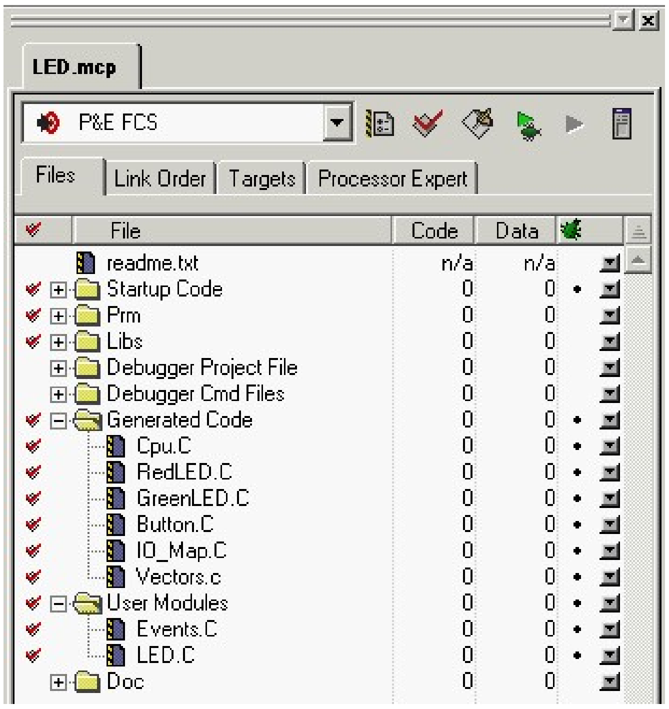
Project: LED1

Module: IO_Map

Current line: 400 Total lines: 1881

Errors: 0 Warnings: 0 Hints: 0

2. Code generation process. This process generates all source files from beans to the "Generated Code" folder in the CodeWarrior project window. The other modules can be found in the "User modules" folder in the CodeWarrior project window. The generated code is inserted only into the selected target in the CodeWarrior project window. See the picture below.



File	Code	Data
readme.txt	n/a	n/a
Startup Code	0	0
Prm	0	0
Libs	0	0
Debugger Project File	0	0
Debugger Cmd Files	0	0
Generated Code	0	0
Cpu.C	0	0
RedLED.C	0	0
GreenLED.C	0	0
Button.C	0	0
IO_Map.C	0	0
Vectors.c	0	0
User Modules	0	0
Events.C	0	0
LED.C	0	0
Doc	0	0

Next Step

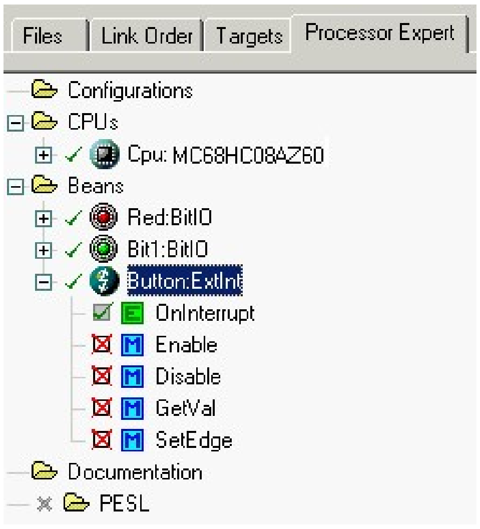
Go to Step 4 - Adding On-Event Code

4.2.4. Tutorial for Freescale HC08 Project 2 Step 4

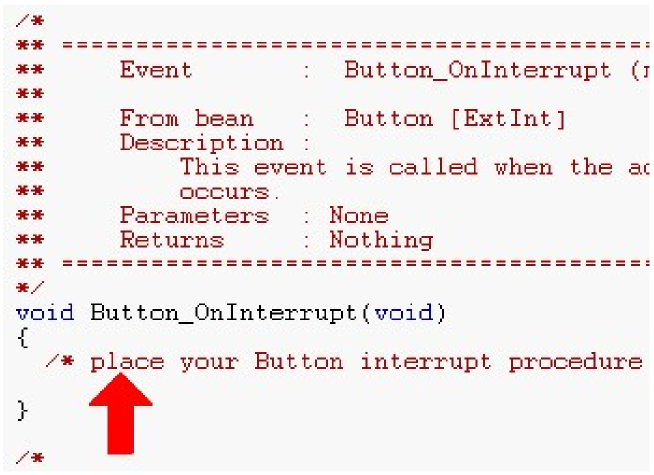
Adding the On-Event Code

1. Switch to the Project panel (Processor expert tab in the CodeWarrior project panel). All the beans (including CPU beans) in the project panel are organized in a tree. You may expand and collapse them by clicking on the plus "+" or minus "-" sign. Bean's events and methods are present as a subnodes of the bean node.
   Note: By double-clicking on any event/method icon, you change its ***enable/disable*** *state (you can do it also in the bean inspector). You need to invoke code generation again to generate code according to the new settings.*

2. Click the "+" sign to expand the Button bean and display its events and methods.



3. Double-click the **OnInterrupt** event from the *Button* bean to open and find out the position of this event in code. See the picture below.



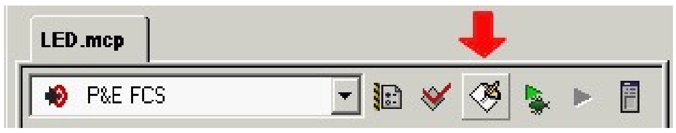
```
/*
** ========================================
**     Event       :  Button_OnInterrupt (
**
**     From bean   :  Button [ExtInt]
**     Description :
**         This event is called when the a
**         occurs.
**     Parameters  :  None
**     Returns     :  Nothing
** ========================================
*/
void Button_OnInterrupt(void)
{
  /* place your Button interrupt procedure

}

/*
```

4. Enter the following lines to the body of the Button_OnInterrupt function:

RedLED_NegVal ();
GreenLED_NegVal ();

The first line is a call of the method NegVal of the bean RedLED. The second line is a call of the method NegVal of the bean GreenLED. Calling syntax of all bean's methods is 'BeanName'_'MethodName'();

5. Finally, to create a binary executable file click on "Make" icon in the CodeWarrior Project Window.



HCS12 Project 1
HCS12 Project 2

4.3. Tutorial Project 1 for Freescale HCS12 Microcontrollers

This simple animated tutorial describes a periodically blinking LED project. The LED is connected to the one pin of the CPU and it is controlled by a periodical timer interrupt. Please follow the instructions and animations showing the actions.

Click here to start the tutorial

4.4. Tutorial Project 2 for Freescale HCS12 Microcontrollers

This tutorial describes a demo project of a simple LED controller. The LED controller has two color LEDs - a red and a green one - and one command button.

How it works

The button sends commands (external interrupts) to CPU through one pin and the CPU switches the red and green LEDs lights on or off. If you press the key you can see that lights of two LEDs have been changed. One of them is switched off and the other one is switched on. At the beginning the green LED is on and the red one is off.

Minimal required hardware design

In the demo application the following components are used:

1. CPU MC9S12DP256 Freescale processor (HCS12 family)
2. Red LED - connected to CPU output pin PB0_ADDR0_DATA0
3. Green LED - connected to CPU output pin PB1_ADDR1_DATA1
4. Button - connected to CPU input pin PH0_KWH0_MISO1

Beans

This simple demo-project uses the following beans:

1. MC9S12DP256 - CPU bean (Freescale HCS12 processor family)
2. BitIO - General 1-bit input/output bean - outputs to LEDs. The LEDs receive 1 bit data which specifies whether the light should switch on or off (value 0 = switch off, value 1 = switch on).
3. ExtInterrupt - External Interrupt bean - interrupt from button. Pressing the button calls an external interrupt which switches the state of the LEDs (on/off).

steps

There are step-by-step instructions how to create this tutorial project. This tutorial goes through the following steps:

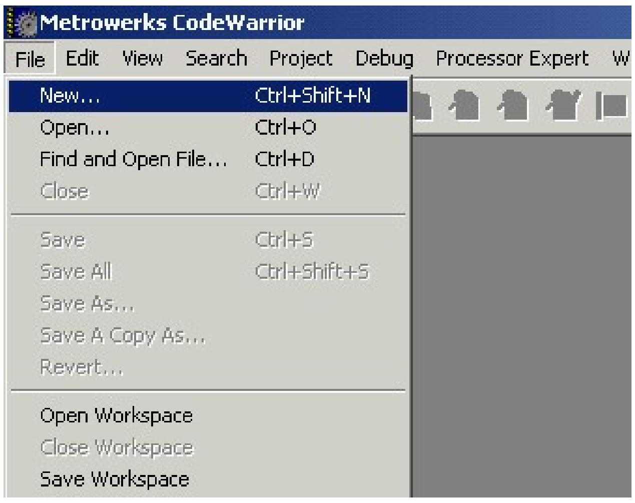
1. Creating a new project
2. Adding beans
3. Generating Code
4. Adding the On-Event Code

Note: This demo project does not care about non-defined states on the output of the key during the process of key pressing. This may result in the fact that state of two LEDs stays apparently unchanged.

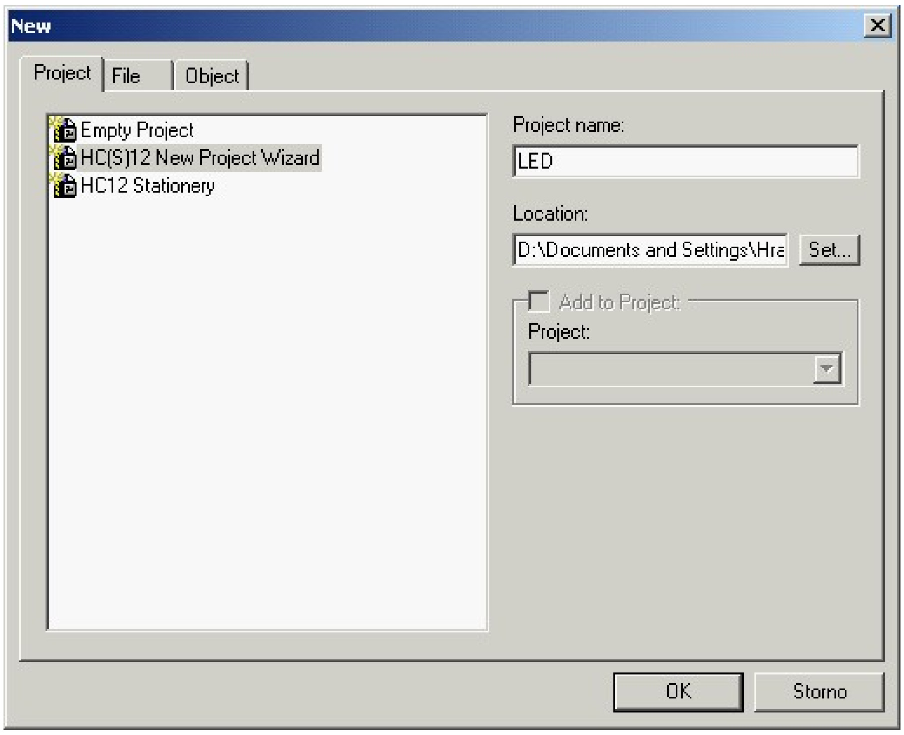
4.4.1. Tutorial for Freescale HCS12 Project 2 Step 1

Creating a New Project

1. Click on the command "New" in the menu "File" in the CodeWarrior Main panel window in order to create a new project.



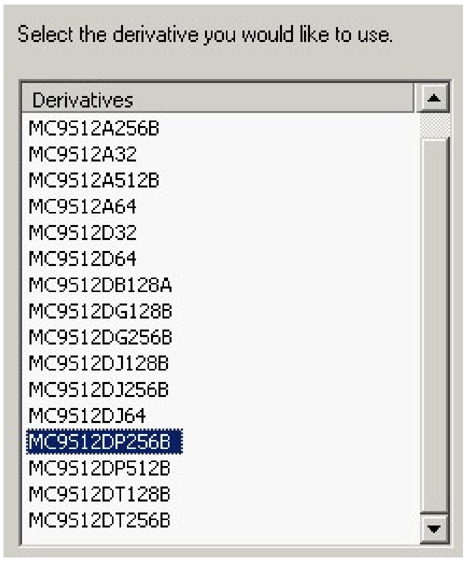
2. Select "HC(S)12 New Project wizard" in the "New" dialog window and enter the name of the project - LED.



Project name: LED

Location: D:\Documents and Settings\Hra

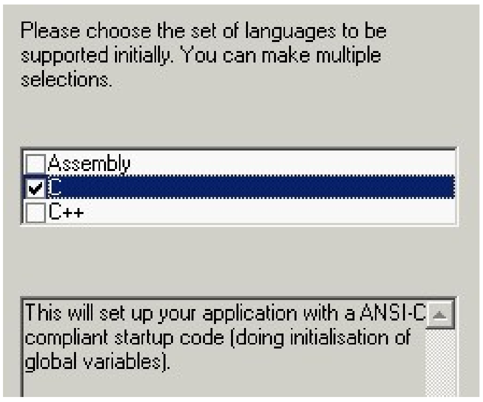
3. In the New Project Wizard window Choose CPU (MC9S12DP256) and click "next" button.



Select the derivative you would like to use.

Derivatives
MC9S12A256B
MC9S12A32
MC9S12A512B
MC9S12A64
MC9S12D32
MC9S12D64
MC9S12DB128A
MC9S12DG128B
MC9S12DG256B
MC9S12DJ128B
MC9S12DJ256B
MC9S12DJ64
MC9S12DP256B
MC9S12DP512B
MC9S12DT128B
MC9S12DT256B

4. Choose "C language" and click "next" button.


Please choose the set of languages to be supported initially. You can make multiple selections.

- [ ] Assembly
- [✓] C
- [ ] C++

This will set up your application with a ANSI-C compliant startup code (doing initialisation of global variables).

5. Choose "yes" in the "Would you like to use Processor Expert dialog" and click "next" button.
6. Choose "no" in the "Do you want to creat a project set up fot PC-lit? dialog" and click "next" button.
7. Choose "none" in the floating point format support selection dialog. click "next" button.
8. Choose "small" memory model. click "next" button.

Now the new empty project is created and ready for adding new beans.

Next step

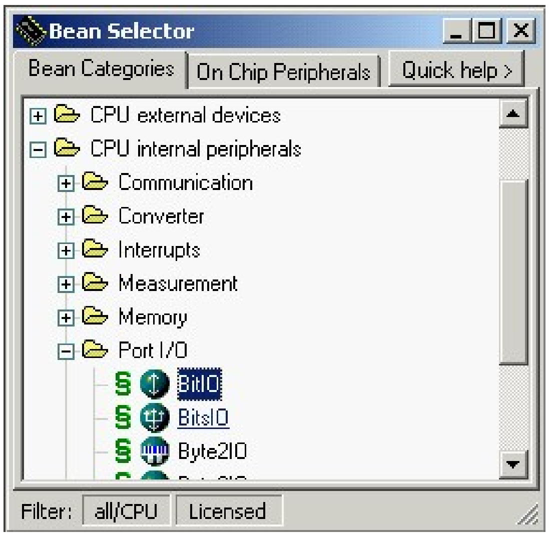
Go to Step 2 - Adding beans to the project.

4.4.2. Tutorial for Freescale HCS12 Project 2 Step 2

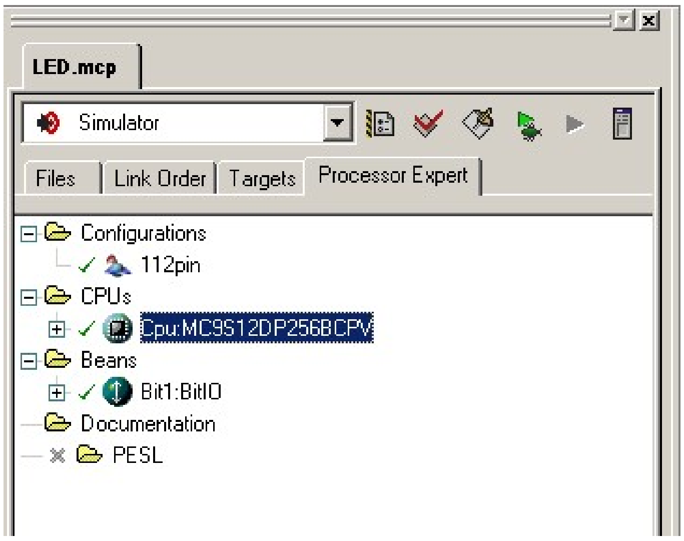
Adding Beans to the Project

In the context of the LED controller, you will add two **BitIO** beans for the Red LED and the Green LED, and an **ExtInt** bean for the Button.

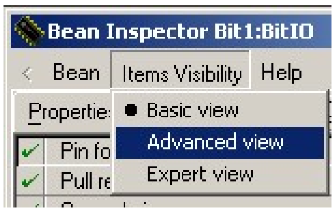
1. If the Bean Selector window is not already opened, open it using menu **Processor Expert | View | Bean Selector**
2. Double click the **BitIO** bean in the folder PortI/O (subfolder of CPU Internal Peripherals folder) in the Bean selector window.



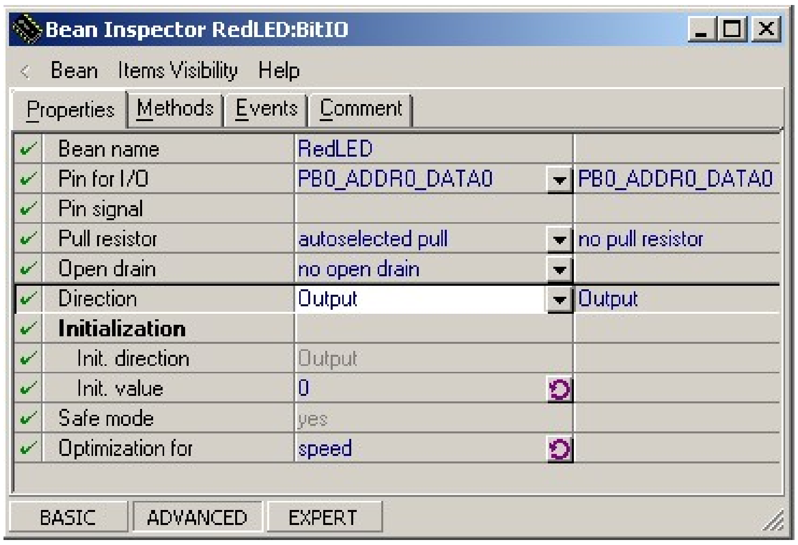
3. If you will be asked whether to enable the bean in all configurations. Choose **Yes**. *(This dialog box could be enabled or disabled in environment options.)*
4. New bean is added to the project. Switch to Processor Expert project tab in codewarrior's project panel. (See picture below). *Don't worry about red exclamation mark beside a new bean. It means that error is present - bean has not been set-up properly yet.*



5. Open the **Bean Inspector** window by double click on the new bean (Bit1:BitIO) in Processor Expert project panel (if the Bean Inspector hasn't been opened automatically). *Automatical opening of the bean inspector is influenced by environment settings (See chapter 2.1.1 Processor Expert Options* for details.)
6. Select **Items visibility | Advanced view** in the pop-up menu of the Bean Inspector window in order to display detailed settings of the bean. It is necessary for the following steps. See picture below.

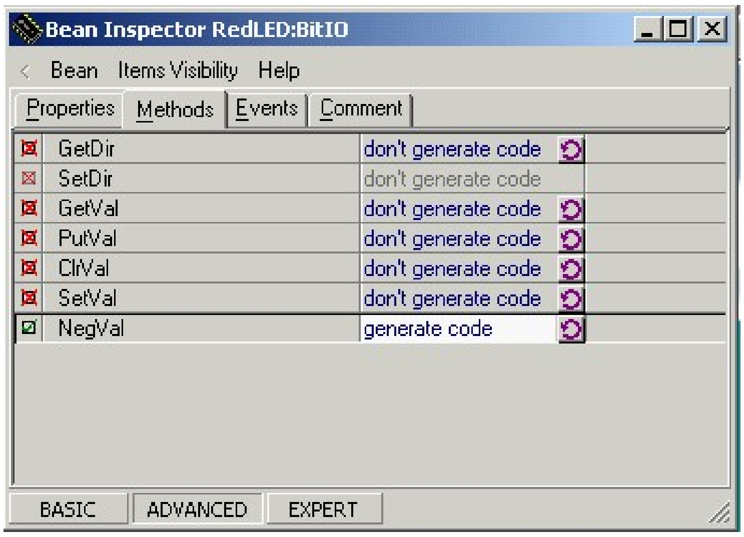


7. Using Bean inspector set the bean properties as follows:
   - **Bean name:** type **RedLED** into the edit box.
   - **Pin for IO:** select **PB0_ADDR0_DATA0**
   - **Direction:** select **Output**



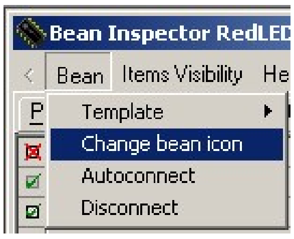
✓	Bean name	RedLED	
✓	Pin for I/O	PB0_ADDR0_DATA0	PB0_ADDR0_DATA0
✓	Pin signal		
✓	Pull resistor	autoselected pull	no pull resistor
✓	Open drain	no open drain	
✓	Direction	Output	Output
✓	**Initialization**		
✓	Init. direction	Output	
✓	Init. value	0	
✓	Safe mode	yes	
✓	Optimization for	speed	

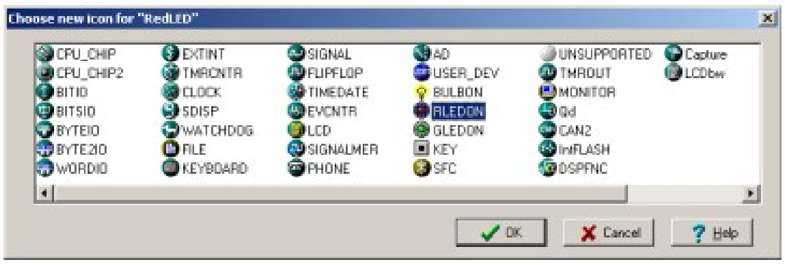
8. To setup generation of **methods** click on the Methods TAB and set all methods to "don't generate" and NegVal method to 'generate code'. See the following picture:



☒	GetDir	don't generate code
☒	SetDir	don't generate code
☒	GetVal	don't generate code
☒	PutVal	don't generate code
☒	ClrVal	don't generate code
☒	SetVal	don't generate code
☑	NegVal	generate code

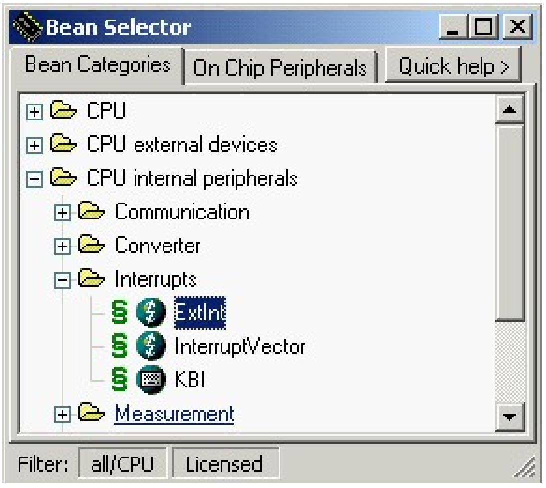
9. click the **Change bean icon** item of the *Bean* menu in order to choose a new icon in the list. Select the **RLEDON** icon and click the **OK** button. See the following pictures:



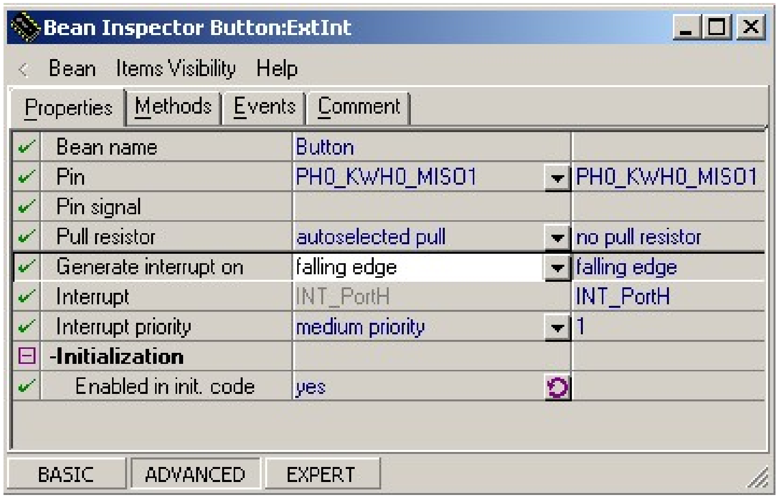


10. Using the procedure previously described for the red LED (steps a,b,c,d) add the **green LED** bean to the project. The difference from the redLED bean is in the bean's properties (pin, name and initial value).
    Set the bean properties as follows:
    - **Bean name:** type **GreenLED** into the edit box.
    - **Pin for IO:** select **PB1_ADDR1_DATA1**
    - **Direction:** select **Output**
    - **Init. value:** select **1**

11. Click on the Methods TAB and switch all methods to "don't generate" and NegVal method to "generate code" using button on the right.
12. Click the **Change bean icon** item of the **Bean** menu in order to choose a new icon. Select the **GLEDON** icon and click the **OK** button.
13. Now is time to add bean handling **the button**. Open the **Bean Selector window** again and double click the **ExtInt** icon on the folder Interrupts in the in order to add the bean to the project. See picture below:



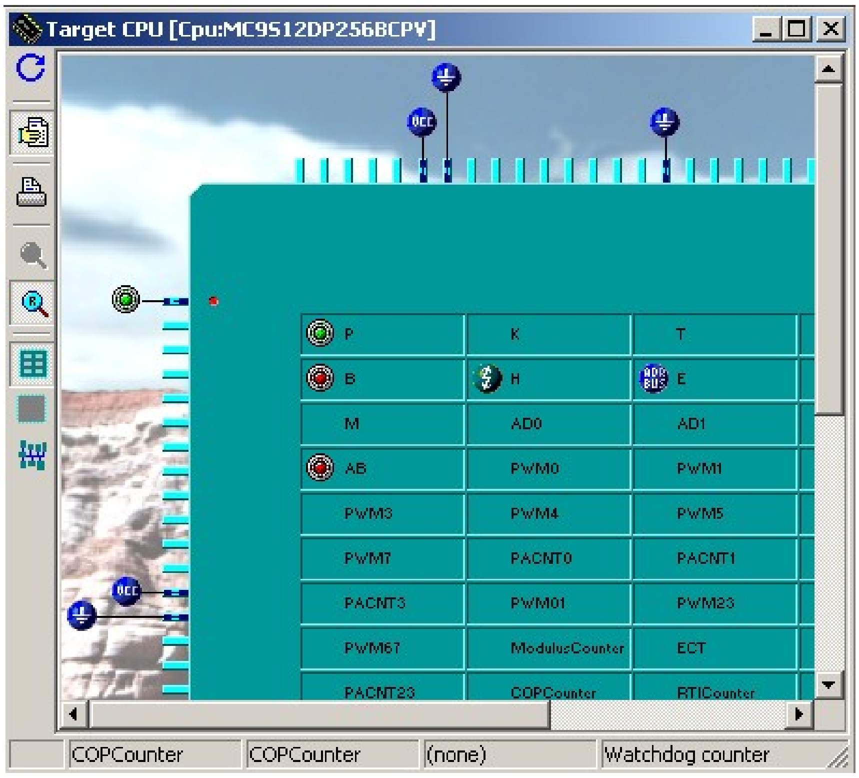
14. Open the **bean inspector** for the new bean (double click it in the project panel) and set the bean properties as follows:
    - **Bean name:** type **Button** into the edit box.
    - **Pin:** select **PH0_KWH0_MISO1**
    - **Generate interrupt on:** click the option in order to display the options. Select the **falling edge**.



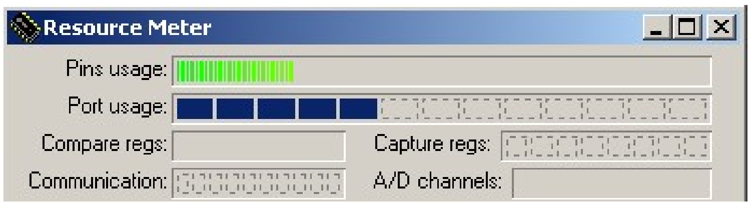
✓	Bean name	Button	
✓	Pin	PH0_KWH0_MISO1	PH0_KWH0_MISO1
✓	Pin signal		
✓	Pull resistor	autoselected pull	no pull resistor
✓	Generate interrupt on	falling edge	falling edge
✓	Interrupt	INT_PortH	INT_PortH
✓	Interrupt priority	medium priority	1
⊟	-Initialization		
✓	Enabled in init. code	yes	

15. Click on the Methods TAB and set **all methods to "don't generate"**.
16. Click the *Change bean icon* item of the *Bean* menu in order to choose a new icon in the list. Select the **KEY** icon and click the **OK** button.

You can see in the Target CPU window which pins of the chip are handled by the beans. You can easily identify LED beans by their specific icons. If the Target CPU window is not opened, use menu command ***Processor Expert | View | Target CPU package***



After adding all beans, click ***Processor Expert | View | Resource Meter*** *in order to open the Resource Meter window and see remaining available resources of the chip.*



Next step

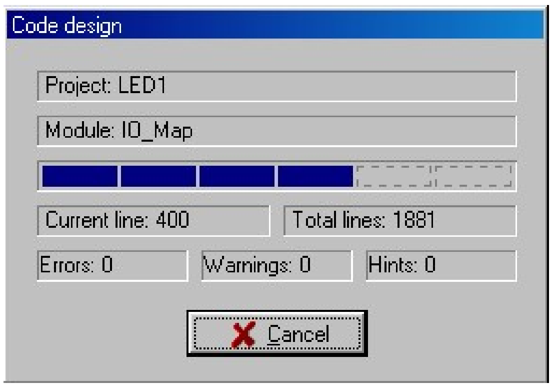
Go to Step 3 - Code Generation.

4.4.3. Tutorial for Freescale HCS12 Project 2 Step 3

Code Generation

1. Click on the command **Processor expert | Generate Code 'Led.mcp'** in the CodeWarrior main menu in order to run the code generation process
   The code generation window shows the current state of code generation.

   Note: *There shouldn't be any errors in the Error window before code generation.*



Code design

Project: LED1

Module: IO_Map

Current line: 400

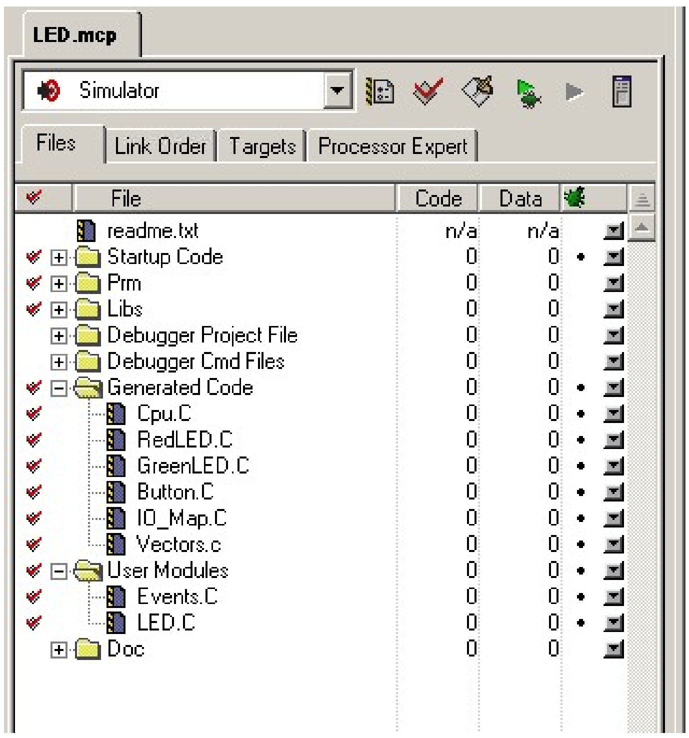
Total lines: 1881

Errors: 0

Warnings: 0

Hints: 0

2. The code generation process generated all source files from beans to the "Generated Code" folder in the CodeWarrior project window. The other modules can be found in the "User modules" folder in the CodeWarrior project window. *The generated code is inserted only into the selected target in the CodeWarrior project window.* See the picture below.



✓	File	Code	Data	
	readme.txt	n/a	n/a	
✓	Startup Code	0	0	•
✓	Prm	0	0	
✓	Libs	0	0	
	Debugger Project File	0	0	
	Debugger Cmd Files	0	0	
✓	Generated Code	0	0	•
✓	Cpu.C	0	0	•
✓	RedLED.C	0	0	•
✓	GreenLED.C	0	0	•
✓	Button.C	0	0	•
✓	IO_Map.C	0	0	•
✓	Vectors.c	0	0	•
✓	User Modules	0	0	•
✓	Events.C	0	0	•
✓	LED.C	0	0	•
	Doc	0	0	

Next Step

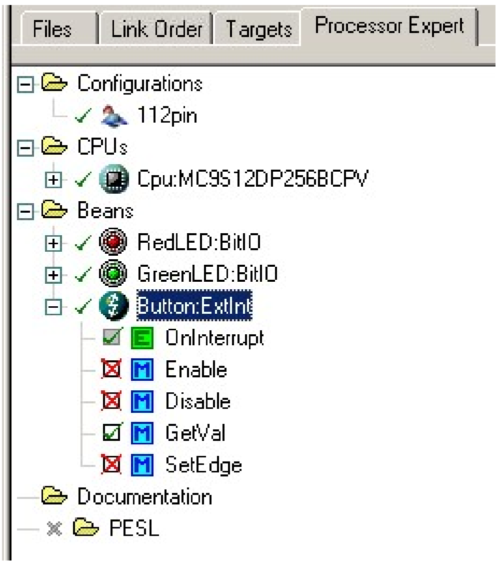
Go to Step 4 - Adding On-Event Code

4.4.4. Tutorial for Freescale HCS12 Project 2 Step 4

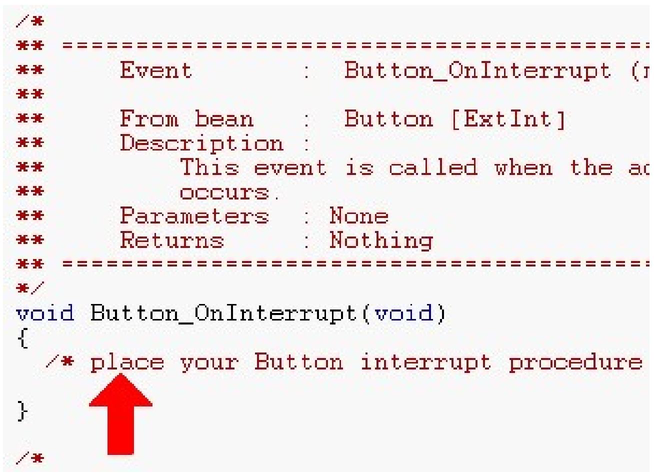
Adding the On-Event Code

1. Switch to the Project panel (Processor expert tab in the CodeWarrior project panel). All the beans (including CPU beans) in the project panel are organized in a tree. You may expand and collapse them by clicking on the plus "+" or minus "-" sign. Bean's events and methods are present as a subnodes of the bean node.
   *Note: By double-clicking on any event/method icon, you change its **enable/disable** state (you can do it also in the bean inspector). You need to invoke code generation again to generate code according to the new settings.*

2. Click the "+" sign to expand the Button bean and display its events and methods.



3. Double-click the **OnInterrupt** event from the *Button* bean to open and find out the position of this event in code. See the picture below.

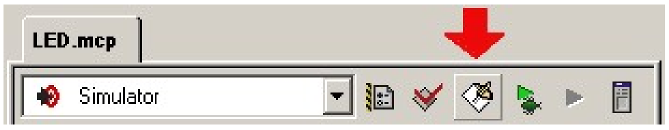


```
/*
** ========================================
**     Event       :  Button_OnInterrupt (
**
**     From bean   :  Button [ExtInt]
**     Description :
**         This event is called when the ac
**         occurs.
**     Parameters  :  None
**     Returns     :  Nothing
** ========================================
*/
void Button_OnInterrupt(void)
{
  /* place your Button interrupt procedure

}

/*
```

4. Enter the following lines to the body of the Button_OnInterrupt function:
   RedLED_NegVal ();
   GreenLED_NegVal ();

   The first line is a call of the method NegVal of the bean RedLED. The second line is a call of the method NegVal of the bean GreenLED. Calling syntax of all bean's methods is 'BeanName'_'MethodName'();

5. Finally, to create a binary executable file click on "Make" icon in the CodeWarrior Project Window.



4.5. Tutorial Project 1 for Freescale 56800/E Microcontrollers

This simple animated tutorial describes a periodically blinking LED project. The LED is connected to one pin of the CPU and it is controlled by a periodical timer interrupt. Please follow the instructions and animations showing the actions.

Note: The project is prepared for the 56F8346 EVM board, but can be simply applied on any other hardware by selecting the different CPU and its resources.

This tutorial is not included in this PDF version of the document.

4.6. Tutorial Project 2 for Freescale 56800 Microcontroller family

This tutorial describes a demo project for a simple LED controller. The LED controls two color LEDs on the EVM (evaluation module) - a red and a green one - and one command button using the EVM's IRQB button.

How it works

The button sends a commands (via an external interrupt) to CPU through one pin and the CPU turns the red and green LEDs lights on or off. If you press the key you can see that lights of two LEDs have been changed. One of them is switched off and the other one is switched on. At the beginning the green LED is on and the red one is off.

Minimal required hardware design

In this demo application the 56F8346 EVM board is used.

Beans

This simple demo-project uses the following beans:

1. M568346E - CPU bean (Freescale 56800 processor family)
2. BitIO - General 1-bit input/output bean - outputs to the LEDs. The LEDs use 1 bit of data specifying whether the light should switch on or off (value 0 = switch off, value 1 = switch on).
3. ExtInterrupt - External Interrupt bean - interrupt from button. Pressing the button calls an external interrupt which switches the state of the LEDs (on/off).

Steps

There are step-by-step instructions how to create this tutorial project.
This tutorial goes through the following steps:

1. Creating a new project
2. Adding beans
3. Code generation
4. Adding the On-Event Code

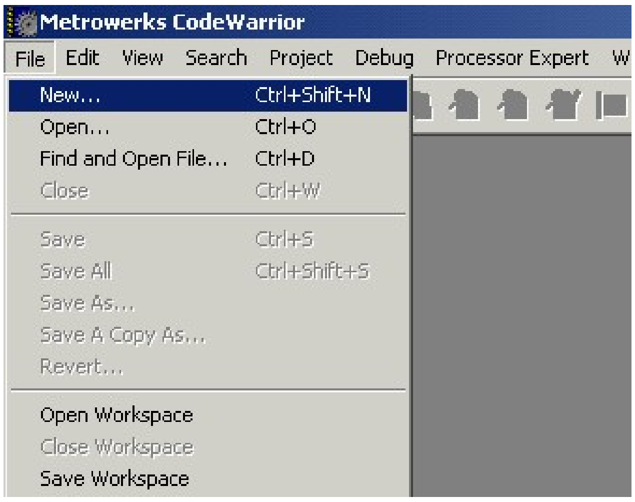
Note: This demo project does not care about non-defined states on the output of the key during the process of key *pressing. This may result in the fact that state of two LEDs stays apparently unchanged.*

4.6.1. Tutorial Freescale 56800 Project 2 Step 1

Creating a New Project

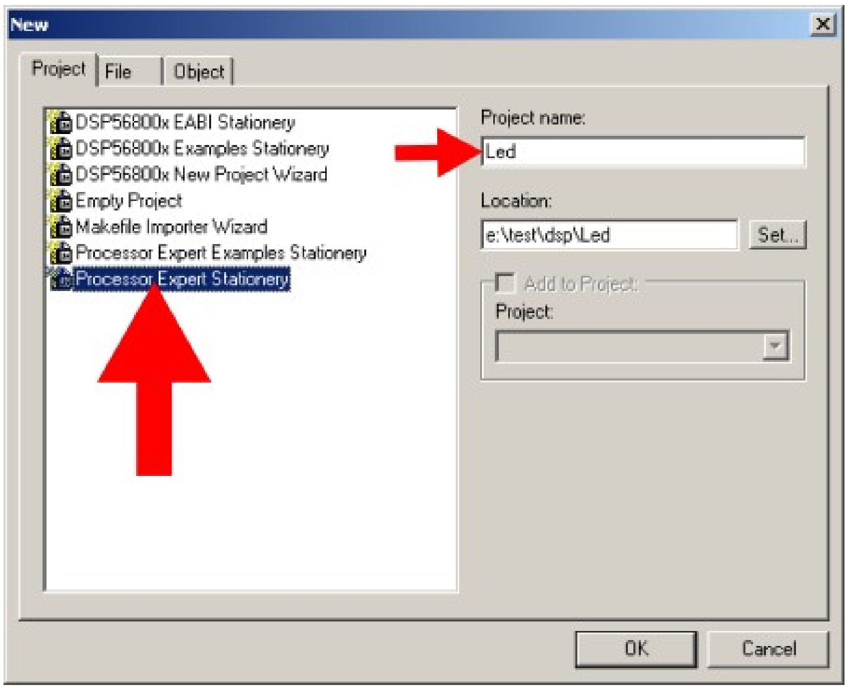
Steps

Click on the command "New" in the menu "File" in the CodeWarrior Main panel window in order to create a new project.

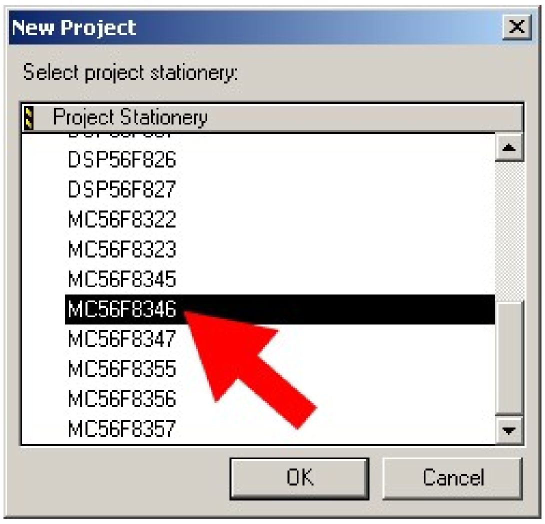


New empty project

Select Processor Expert Stationery in the "New" dialog window and enter name of the project - LED.


New
Project File Object
DSP56800x EABI Stationery
DSP56800x Examples Stationery
DSP56800x New Project Wizard
Empty Project
Makefile Importer Wizard
Processor Expert Examples Stationery
Processor Expert Stationery
Project name:
Led
Location:
e:\test\dsp\Led
Set...
Add to Project:
Project:
OK
Cancel

Select CPU MC56F8346 which will be used in our project.



Next step

Go to Step 2 - Adding beans to the project.

4.6.2. Tutorial Freescale 56800 Project 2 Step 2

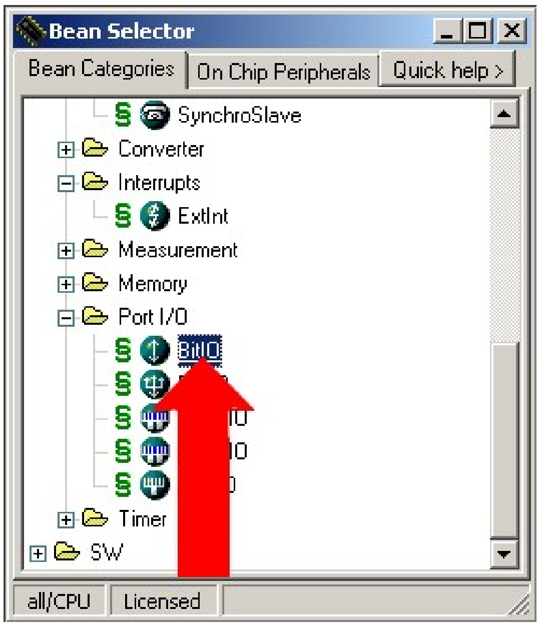
Adding Beans to the Project

Task

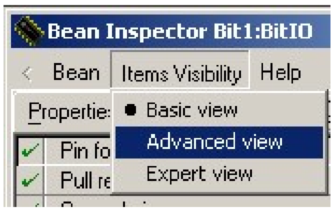
In this step you will add, set and connect the project's beans.
In the context of the LED controller, you will add two **BitIO** beans, one for the Red LED and the second one for the Green LED, and an **ExtInt** bean for the IRQB Button.

Red LED

Double click the **BitIO** bean in the folder PortI/O in the Bean selector window.

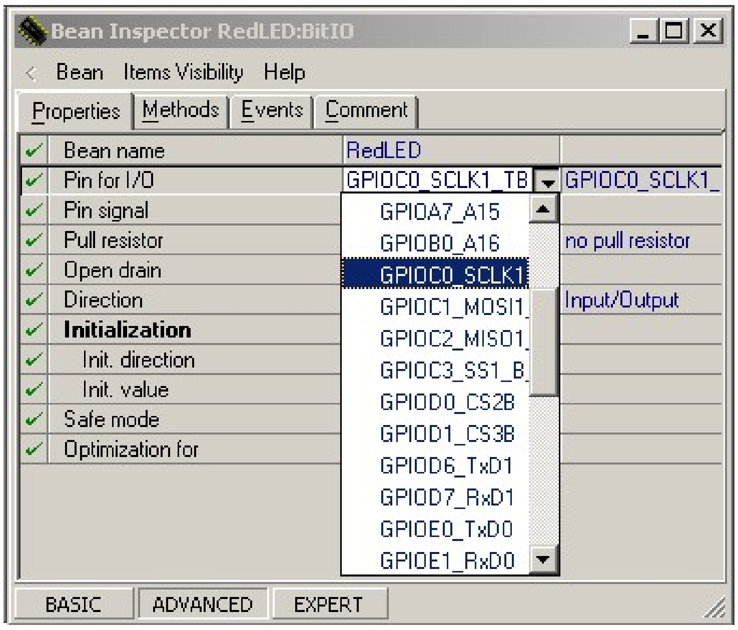


Select "View | Advanced view" in the pop-up menu of the Bean Inspector window in order to display detailed settings of the bean. This is necessary for the following steps. See picture below.



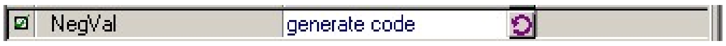
For the RedLED, set the properties of the bean as follows:

- **Bean name :** type **RedLED** into the edit box.
- **Pin for IO:** select **GPIOC0_SCLK1**
- **Direction**: select **Output**



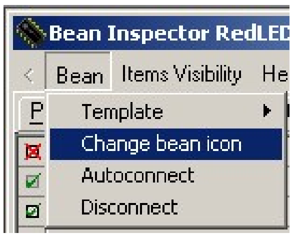
Bean name	RedLED	
Pin for I/O	GPIOC0_SCLK1_TB	GPIOC0_SCLK1_
Pin signal		
Pull resistor		no pull resistor
Open drain		
Direction		Input/Output
Initialization		
Init. direction		
Init. value		
Safe mode		
Optimization for		

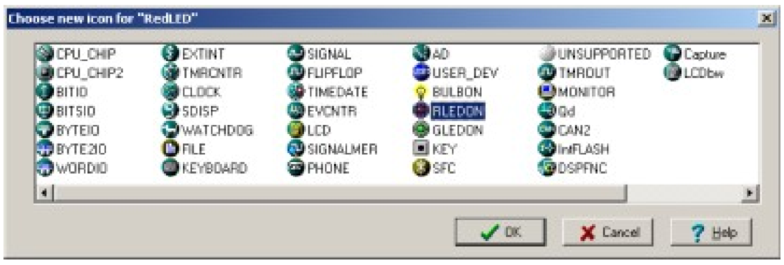
Methods: click on the Methods TAB and set NegVal method to "generate code". *We use the NegVal (Negate Value) method to toggle the value of the GPIO pin, thus changing the LED connected to the pin from on to off or vice versa.*



NegVal generate code

Bean icon: click the *Change bean icon* item of the *Bean* menu in order to choose a new icon in the list. Select the **RLEDON** icon and click the **OK** button. *Different icons allows the +user to easily distinguish beans, especially in the large projects.*





Green LED

Using the procedure previously described for the Red LED, add the Green LED bean to the project.

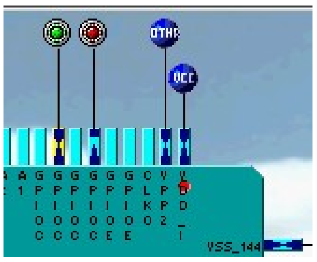
Properties: For the GreenLED, set the bean as follows:

- **Bean name:** type **GreenLED** into the edit box.
- **Pin for IO:**select **GPIOC2_MISO1**
- **Direction:** select **Output**
- **Init. value:** select **1** to turn on the LED initially

Methods: click on the Methods TAB and set NegVal method "generate code". See the following picture:

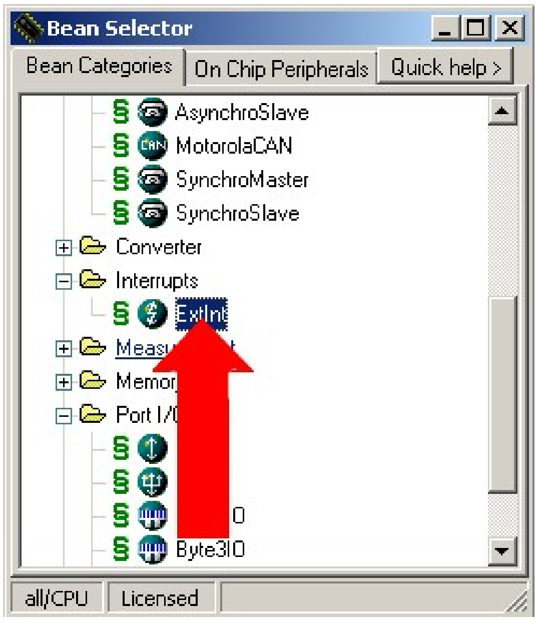
Bean icon: click the *Change bean icon* item of the *Bean* menu in order to choose a new icon. Select the **GLEDON** icon and click the **OK** button. See the following pictures:

You can see in the Target CPU window where the GreenLED bean and RedLED bean are connected (the appropriate pins is colored in yellow). If the Target CPU window is not already opened, choose **Processor Expert|View|Target CPU**



Button

Double click the **ExtInt** icon on the folder Interrupts in the Bean Selector window in order to add the bean to the project. See picture below:

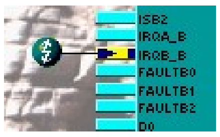


For the button, set the bean as follows:

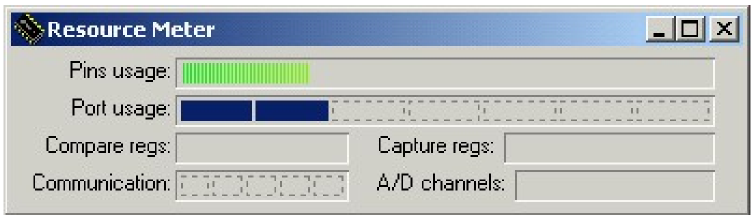
- **Bean name:** type **Button** into the edit box.
- **Pin:** select **IRQB_B**
- **Generate interrupt on:** click the option in order to display the options. Select the **falling edge**.

Methods: click on the Methods TAB and set all methods to "don't generate ".

You can see in the Target CPU window where the Button bean is connected (the pin is colored in yellow). If the Target CPU window is not opened, click **Processor Expert | View | Target CPU**



Note: *After adding all beans, you can click Processor Expert|View|Resource Meter in order to open the Resource Meter window and see remaining available resources of the chip.*



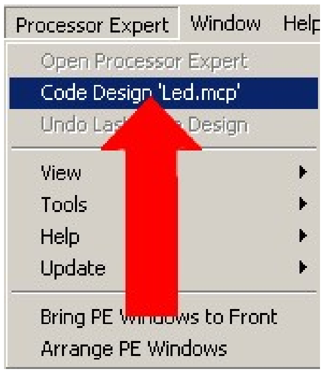
Next step

Go to Step 3 - Code Generation

4.6.3. Tutorial Freescale 56800 Project 2 Step 3

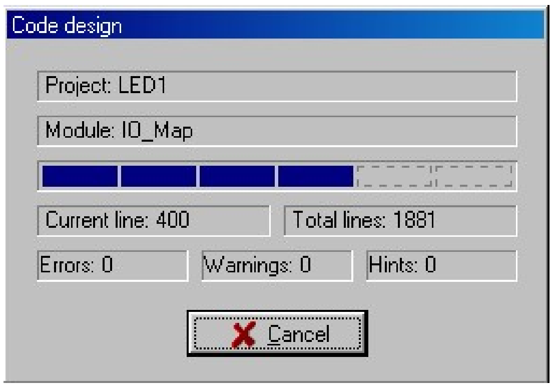
Code Generation

Click on the command "Generate Code {project}" in the CodeWarrior main menu in order to run the code generation process.



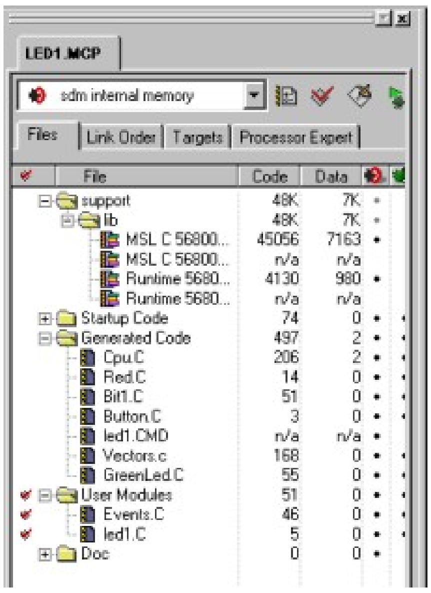
The code generation process window shows you the running state of code generation.

Note: *There shouldn't be any errors in the Error window before code generation.*



Project: LED1
Module: IO_Map
Current line: 400 | Total lines: 1881
Errors: 0 | Warnings: 0 | Hints: 0

Code generation process. The code generation process generates all source code from beans and stores files in the "Generated Code" folder in the CodeWarrior project window. The other modules can be found in the "User modules" folder in the CodeWarrior project window. The generated code is inserted only into the selected target in the 'Files' tab of the CodeWarrior project window. See the picture below.



✓	File	Code	Data		
	support	48K	7K	•	
	lib	48K	7K	•	
	MSL C 56800...	45056	7163	•	
	MSL C 56800...	n/a	n/a		
	Runtime 5680...	4130	980	•	
	Runtime 5680...	n/a	n/a		
	Startup Code	74	0	•	•
	Generated Code	497	2	•	•
	Cpu.C	206	2	•	•
	Red.C	14	0	•	•
	Bit1.C	51	0	•	•
	Button.C	3	0	•	•
	led1.CMD	n/a	n/a	•	
	Vectors.c	168	0	•	•
	GreenLed.C	55	0	•	•
✓	User Modules	51	0	•	•
✓	Events.C	46	0	•	•
✓	led1.C	5	0	•	•
	Doc	0	0	•	

Next Step

Go to Step 4 - Adding On-Event Code

4.6.4. Tutorial Freescale 56800 Project 2 Step 4

Adding the On-Event Code

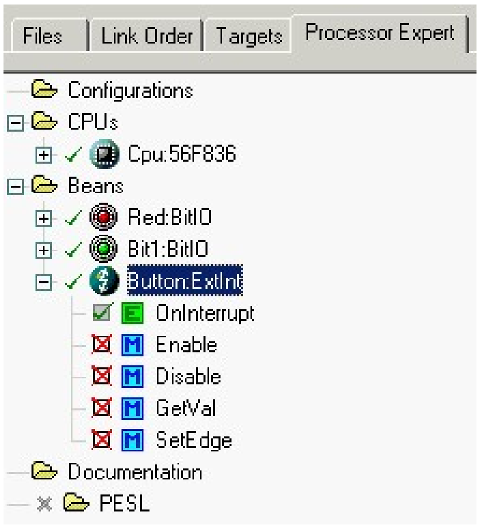
Task

This step covers how to complete the project by writing the Interrupt Service Routine (ISR) that services the IRQB button's external interrupt. This code is added to the file Events.c, which holds all user code for events used in the project.

View of bean events and method

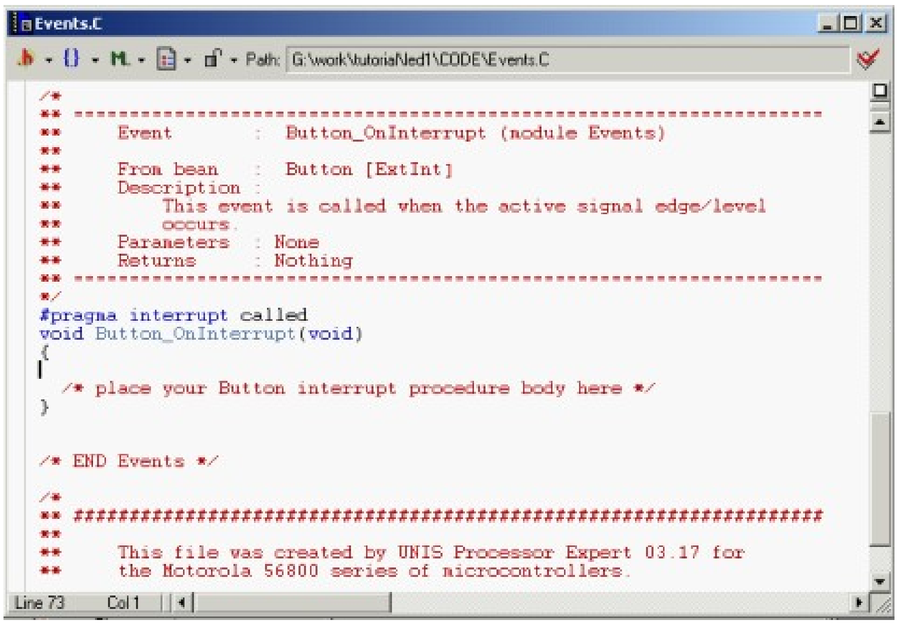
All the beans (including CPU beans) in the project panel are organized in a tree. You may expand and collapse them by clicking on the plus "+" or minus "-" sign. There are all bean events and methods in the tree under bean.

Note: *By double-clicking on any event/method icon, you change its enable/disable state (you can do it also in the bean inspector). Then, click the code generation option to code design the code again.*
Click the "+" sign to expand the Button bean and display its events and methods.



After code generation, a double-click on any event/method name opens the file editor/viewer at the position of the event/method code.

Double-click the **OnInterrupt** event from the *Button* bean to open and find out the location of this event in code (*Events.c file*). See the picture below.



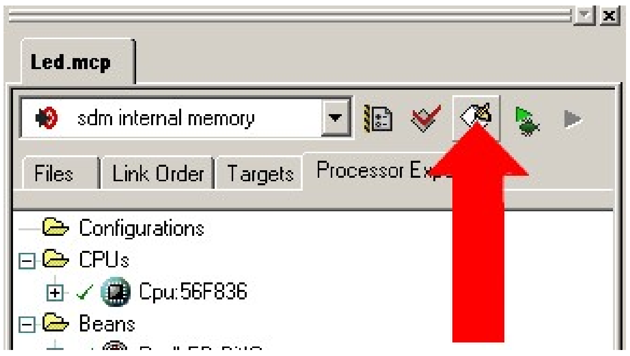
```
/*
** ===================================================================
**     Event       :  Button_OnInterrupt (module Events)
**
**     From bean   :  Button [ExtInt]
**     Description :
**         This event is called when the active signal edge/level
**         occurs.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
#pragma interrupt called
void Button_OnInterrupt(void)
{

  /* place your Button interrupt procedure body here */
}


/* END Events */

/*
** ###################################################################
**
**     This file was created by UNIS Processor Expert 03.17 for
**     the Motorola 56800 series of microcontrollers.
```

Enter the following commands into the Events.c file:

RedLED_NegVal (); /*toggle state of Red LED*/
GreenLED_NegVal (); /*toggle state of Green LED*/

The first line is calling of the method NegVal of the bean RedLED. The second line is calling of the method NegVal of the bean GreenLED. Calling syntax of all methods is 'BeanName'_'MethodName'();

To create a binary executable file click on "Make" icon in the CodeWarrior Project Window. See picture below.



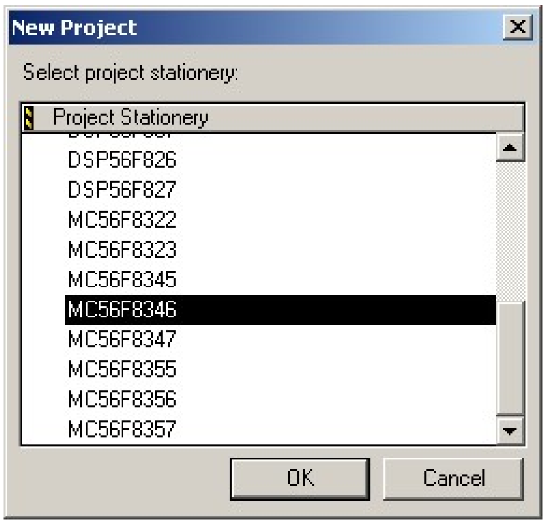
Project is finished. Now you can upload it to the board and execute it.

4.7. Tutorial Project 3 for Freescale 568000 Microcontroller family

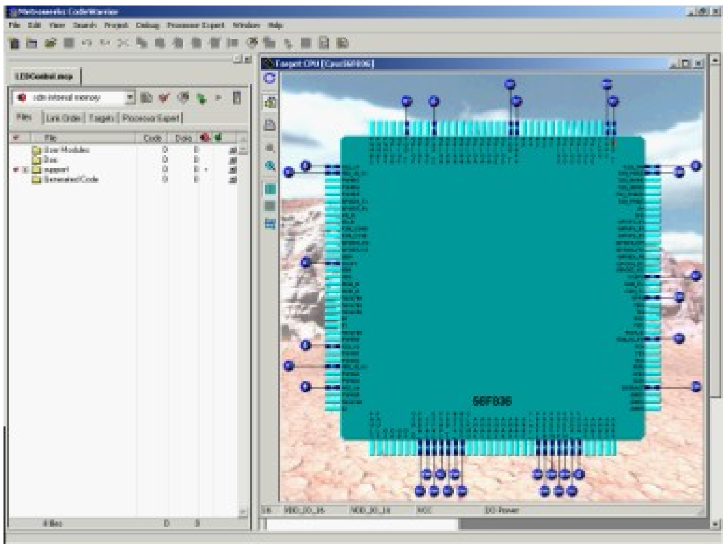
This tutorial exercise creates a project that flashes the LEDs of a DSP568346E EVM (evaluation module). This project is more complicated than the previous ones. If you are a beginner try Project 1 first.

Follow these steps:

1. Create a 56836E project, using C with Processor Expert stationery.
   a. Start the CodeWarrior IDE, if it is not started already.
   b. From the main-window menu bar, select File > New. The New window appears.
   c. In the Project page, select (highlight) Processor Expert Stationery.
   d. In the Project name text box, enter a name for the project, such as LEDcontrol.
   e. Click the OK button. The New Project window replaces the New window.
   f. In the Project Stationery list, select the MC56F8346 entry.

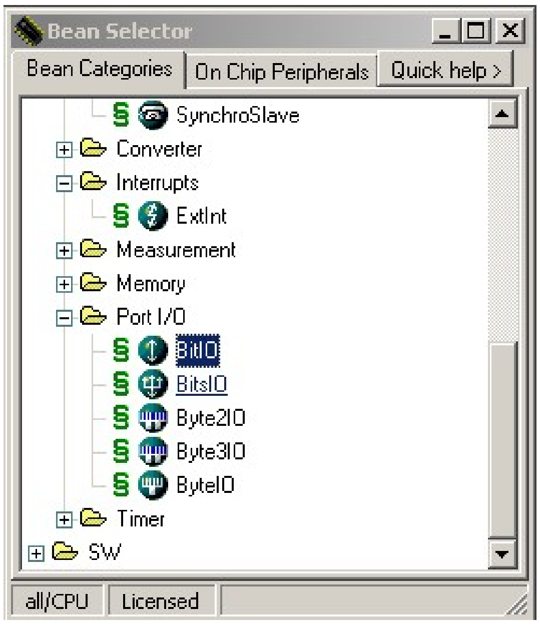


   g. Click the OK button. The IDE:
      - Opens the project window, docking it the left of the main window. This project window includes a Processor Expert page.
      - Opens the Target CPU window, as the following picture shows. This window shows the the CPU package and peripherals view.

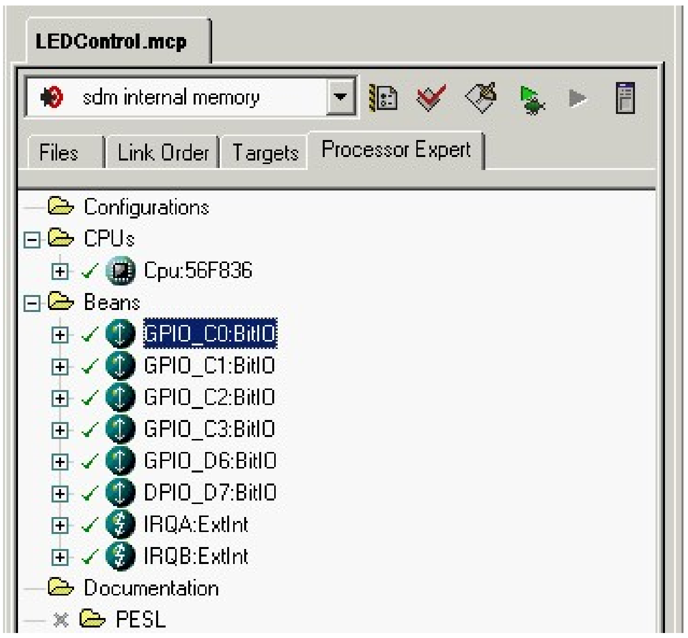


- Opens the Bean Selector window, behind the Target CPU window.

2. Select the sdm external memory target to use small data memory model and external memory.
   a. Click the project window's Targets tab. The Targets page moves to the front of the window.
   b. Click the target icon of the sdm external memory entry. The black arrow symbol moves to this icon, confirming your selection.
3. Add six BitIO beans to the project.
   a. Click the project window's Processor Expert tab. The Processor Expert page moves to the front of the window.
   b. Make the Bean Selector window visible:
      - Minimize the Target CPU window.
      - Select Processor Expert > View > Bean Selector, from the main-window menu bar.
   c. In the Bean Categories page, expand the entry CPU internal peripherals.
   d. Expand the subentry Port I/O.
   e. Double-click the BitIO bean name six times. (folloging figure depicts this bean selection.) The IDE adds these beans to your project; new bean icons appear in the project window's Processor Expert page.



4. Add two ExtInt beans to the project.
   a. In the Bean Categories page of the Bean Selector window, expand the Interrupts subentry.
   b. Double-click the ExtInt bean name two times. The IDE adds these beans to your project; new bean icons appear in the Processor Expert page.
   c. You may close the Bean Inspector window.
5. Rename the eight beans GPIO_C0 — GPIO_C3, GPIO_D6, GPIO_D7, IRQA, and IRQB.
   a. In the project window's Processor Expert page, right-click the name of the first BitIO bean. A context menu appears.
   b. Select Rename Bean. A change box appears around the bean name.
   c. Type the new name GPIO_C0, then press the Enter key. The list shows the new name; as the following shows, this name still ends with BitIO.
   d. Repeat substeps a, b, and c for each of the other BitIO beans, renaming them GPIO_C1, GPIO_C2, GPIO_C3, GPIO_D6, and GPIO_D7.
   e. Repeat substeps a, b, and c for the two ExtInt beans, renaming them IRQA and IRQB. Now the project panel should look like the following figure.



Configurations
CPUs
Cpu:56F836
Beans
GPIO_C0:BitIO
GPIO_C1:BitIO
GPIO_C2:BitIO
GPIO_C3:BitIO
GPIO_D6:BitIO
DPIO_D7:BitIO
IRQA:ExtInt
IRQB:ExtInt
Documentation
PESL

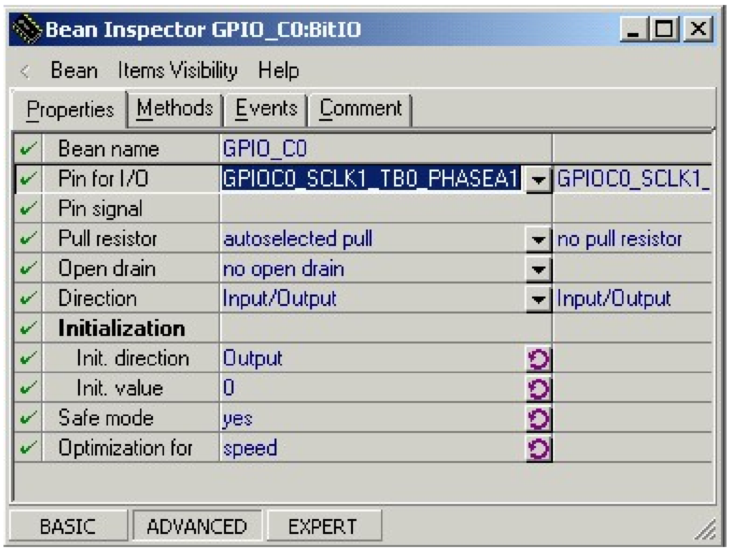
6. Update pin associations for each bean.

   a. In the Processor Expert page, double-click the bean name GPIO_C0. The Bean Inspector window opens, displaying information for this bean.

   b. Use standard window controls to make the middle column of the Properties page about 2 inches wide.

   c. In the Pin for I/O line, click the triangle symbol of the middle-column list box. The list box opens.

   d. Use this list box to select GPIOC0_SCLK1_TB0_PHASEA1. Followin fFigure depicts this selection.



✓	Bean name	GPIO_C0	
✓	Pin for I/O	GPIOC0_SCLK1_TB0_PHASEA1	GPIOC0_SCLK1_
✓	Pin signal		
✓	Pull resistor	autoselected pull	no pull resistor
✓	Open drain	no open drain	
✓	Direction	Input/Output	Input/Output
✓	**Initialization**		
✓	Init. direction	Output	
✓	Init. value	0	
✓	Safe mode	yes	
✓	Optimization for	speed	

   e. In the project window's Processor Expert page, select the bean name GPIO_C1. The Bean Inspector information changes accordingly

   f. Use the Pin for I/O middle-column list box to select GPIOC1_MOSI1_TB1_PHASEB1

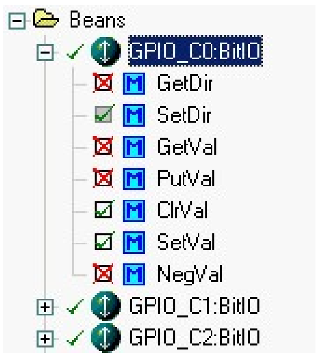
   g. Repeat substeps e and f, for bean GPIO_C2, to change its associated pin to GPIOC2_MISO1_TB2_INDEX1

h. Repeat substeps e and f, for bean GPIO_C3, to change its associated pin to GPIOC3_SSA_B_TB3_HOME1
i. Repeat substeps e and f, for bean GPIO_D6, to change its associated pin to GPIOD6_TxD1
j. Repeat substeps e and f, for bean GPIO_D7, to change its associated pin to GPIOD7_RxD1
k. In the project window's Processor Expert page, select the bean name IRQA

The Bean Inspector information changes accordingly

l. Use the Pin middle-column list box to select IRQA_B
m. Repeat substeps k and l, for bean IRQB, to change its associated pin to IRQB_B
n. You may close the Bean Inspector window

7. Enable BitIO SetDir, ClrVal, and SetVal functions.
   a. In the Processor Expert page, click the plus-sign control for the GPIO_C0 bean. The function list expands: red X symbols indicate disabled functions, green check symbols indicate enabled functions
   b. Double-click function symbols as necessary, so that only SetDir, ClrVal, and SetVal have green checks. (Following figure shows this configuration.)



   c. Click the GPIO_C0 minus-sign control. The function list collapses
   d. Repeat substeps a, b, and c for beans GPIO_C1, GPIO_C2, GPIO_C3, GPIO_D6, and GPIO_D7

8. Enable ExtInt OnInterrupt, GetVal functions.
   a. In the Processor Expert page, click the plus-sign control for the IRQA bean. The function list expands.
   b. Double-click function symbols as necessary, so that only OnInterrupt and GetVal have green check symbols.
   c. Click the IRQA minus-sign control. The function list collapses.
   d. Repeat substeps a, b, and c for bean IRQB.

9. Generate project code.
   a. From the main-window menu bar, select Processor Expert > Generate Code ‘LEDcontrol.mcp.' (This selection shows the actual name of your project.) The IDE and PEI generate several new files for your project.
   b. You may close all windows except the project window.

10. Update file Events.c.
   a. Click the project window's Files tab. The Files page moves to the front of the window.
   b. Expand the User Modules folder.
   c. Double-click filename Events.c. An editor window opens, displaying this file's text. (To view

content of the file events.c click here)

d. Find the line IRQB_OnInterrupt().

e. Above this line, enter the new line extern short IRQB_On;.

f. Inside IRQB_OnInterrupt(), enter the new line IRQB_On ^= 1;.

g. Find the line IRQA_OnInterrupt().

h. Above this line, enter the new line extern short IRQA_On;.

i. Inside IRQA_OnInterrupt(), enter the new line IRQA_On ^= 1;.

j. Save and close file Events.c.

11. Update file LEDcontrol.c.

a. In the project window's Files page, double-click filename LEDcontrol.c (or the actual .c filename of your project). An editor window opens, displaying this file's text.

b. Add custom code, to utilize the beans. (To view content of the file LEDcontrol.c click here, you can use clipboard to transfer the source code to CodeWarrior).

c. Save and close the file.

12. Build and debug the project.

a. From the main-window menu bar, select Project > Make. The IDE compiles and links your project, generating executable code.

b. Debug your project, as you would any other CodeWarrior project.

Second Processor Expert tutorial exercise is complete. Downloading this code to a DSP56836E development board should make the board LEDs flash in a distinctive pattern.

4.7.1. Listing of the File events.c

Manually written code is marked **bold**. Rest of the file is generated by Processor Expert.

```
/*
** ############################################################ **
** Filename : Events. C
**
** Project : LEDcontrol
**
** Processor : DSP56F836
**
** Beantype : Events
**
** Version : Driver 01.00
**
** Compiler : Metrowerks DSP C Compiler
**
** Date/ Time : 3/ 24/ 2003, 1: 18 PM
**
** Abstract :
**
** This is user's event module.
** Put your event handler code here.
**
```

```
** Settings :
**
**
** Contents :
**
** IRQB_ OnInterrupt -void IRQB_ OnInterrupt( void);
** IRQA_ OnInterrupt -void IRQA_ OnInterrupt( void);
**
**
** (c) Copyright UNIS, spol. s r. o. 1997-2002
**
** UNIS, spol. s r. o.
** Jundrovska 33
** 624 00 Brno
** Czech Republic
**
** http : www. processorexpert. com
** mail : info@ processorexpert. com 32
**
** ###########################################################
*/
/* MODULE Events */


/* Including used modules for compilling procedure*/
#include "Cpu. h"
#include "Events. h"
#include "GPIO_ C0. h"
#include "GPIO_ C1. h"
#include "GPIO_ C2. h"
#include "GPIO_ C3. h"
#include "GPIO_ D6. h"
#include "GPIO_ D7. h"
#include "IRQA. h"
#include "IRQB. h"


/* Include shared modules, which are used for whole project*/
#include "PE_ Types. h"
#include "PE_ Error. h"
#include "PE_ Const. h"
#include "IO_ Map. h"


/*
** ===========================================================
** Event : IRQB_ OnInterrupt (module Events)
**
** From bean : IRQB [ExtInt]
** Description :
** This event is called when the active signal edge/ level
** occurs.
** Parameters : None
** Returns : Nothing
** ===========================================================
*/
#pragma interrupt called
extern short IRQB_ On;
void IRQB_ OnInterrupt(void)
{
  IRQB_ On ^= 1;
  /* place your IRQB interrupt procedure body here */
}


/*
** ===========================================================
** Event : IRQA_ OnInterrupt (module Events)
**
** From bean : IRQA [ExtInt]
** Description :
** This event is called when the active signal edge/ level
** occurs.
** Parameters : None
** Returns : Nothing
** ==========================================================
*/
#pragma interrupt called
extern short IRQA_ On;
void IRQA_ OnInterrupt(void)
{

  IRQA_ On ^= 1;

  /* place your IRQA interrupt procedure body here */
}


/* END Events */
/*
** ##########################################################
**
** This file was created by UNIS Processor Expert 03.15 for
** the Freescale DSP56x series of microcontrollers.
**
** ############################################################
```

4.7.2. Listing of the file LEDcontrol.c

Manually written code is marked **bold**. Content of the main module.

```
/*
** ############################################################
** Filename : LEDcontrol. C
**
** Project : LEDcontrol
**
** Processor : 56F8346
**
** Version : Driver 01.00
**
** Compiler : Metrowerks DSP C Compiler
**
** Date/ Time : 3/ 24/ 2003, 1: 18 PM
**
** Abstract :
**
** Main module.
** Here is to be placed user's code.
**
** Settings :
**
**
** Contents :
**
** No public methods
**
**
** (c) Copyright UNIS, spol. s r. o. 1997-2002
**
** UNIS, spol. s r. o.
** Jundrovska 33
** 624 00 Brno
** Czech Republic
**
** http : www. processorexpert. com
** mail : info@ processorexpert. com
**
** ###########################################################
*/
/* MODULE LEDcontrol */


/* Including used modules for compilling procedure */
#include "Cpu. h"
#include "Events. h"
```

```
#include "GPIO_ C0. h"
#include "GPIO_ C1. h"
#include "GPIO_ C2. h"
#include "GPIO_ C3. h"
#include "GPIO_ D6. h"
#include "GPIO_ D7. h"
#include "IRQA. h"
#include "IRQB. h"
/* Include shared modules, which are used for whole project */
#include "PE_ Types. h"
#include "PE_ Error. h"
#include "PE_ Const. h"
#include "IO_ Map. h"



/*
* Application Description:
* LED program for the 56836 EVM.
*
* Pattern: "Count" from 0 to 63, using LEDs to represent the bits of
the number.
*
* Pressing the IRQA button flips LED order: commands that previously
went to LED1 go to LED6, and so forth.
* Pressing the IRQB button reverses the enabled/ disabled LED states.
*
*/


/* global used as bitfield, to remember currently active bits, used to
* enable/ disable all LEDs. */
long num = 0;
short IRQA_ On, IRQB_ On;


/* simple loop makes LED changes visible to the eye */
void wait( int);
voide wait( int count)
{
    int i;
    for (i= 0; i < count; ++ i);
}


/* set the given LED */
void setLED( int);
void setLED( int num)
{
    if (! IRQA_ On)
    {
        num = 7-num;
    }
    if (! IRQB_ On)
    {
        switch (num)
        {
            case 1: GPIO_ C0_ ClrVal(); break;
            case 2: GPIO_ C1_ ClrVal(); break:
            case 3: GPIO_ C2_ ClrVal(); break;
            case 4: GPIO_ C3_ ClrVal(); break;
            case 5: GPIO_ D6_ ClrVal(); break;
            case 6: GPIO_ D7_ ClrVal(); break;
        }
    }
    else
    {
        switch (num)
        {
            case 1: GPIO_ C0_ SetVal(); break;
            case 2: GPIO_ C1_ SetVal(); break;
            case 3: GPIO_ C2_ SetVal(); break;
            case 4: GPIO_ C3_ SetVal(); break;
            case 5: GPIO_ D6_ SetVal(); break;
            case 6: GPIO_ D7_ SetVal(); break;
        }
    }
}


/* clear the given LED */
void clrLED( int);
void clrLED( int num)
{
    if (! IRQA_ On)
    {
        num = 7-num;
    }
    if (IRQB_ On)
    {
        switch (num)
        {
            case 1: GPIO_ C0_ ClrVal(); break;
            case 2: GPIO_ C1_ ClrVal(); break;
            case 3: GPIO_ C2_ ClrVal(): break;
            case 4: GPIO_ C3_ ClrVal(); break;
            case 5: GPIO_ D6_ ClrVal(); break;
            case 6: GPIO_ D7_ ClrVal(); break;
        }
    }
    else
    {
        switch (num)
        {
            case 1: GPIO_ C0_ SetVal(); break;
            case 2: GPIO_ C1_ SetVal(); break;
            case 3: GPIO_ C2_ SetVal(); break;
            case 4: GPIO_ C3_ SetVal(); break;
            case 5: GPIO_ D6_ SetVal(); break;
            case 6: GPIO_ D7_ SetVal(); break;
        }
    }
}


#define CLEARLEDS showNumberWithLEDs( 0)
/* method to set each LED status to reflect the given number/ bitfield */
void shwNumberWithLEDs( long);
void showNumberWithLEDs( long num)
{
    int i;
    for (i= 0; i < 6; ++ i)
    {
        if (( num >> i)  1<br/>            setLED( i+ 1)
        else
            clrLED( i+ 1);
    }
}


/* Pattern: "Count" from 0 to 63 in binary using LEDs to represent bits
of the current number. 1 = enabled LED, 0 = disabled LED. */
void pattern();
void pattern()
{
    long i;
    int j;


    for (i= 0; i<= 0b111111; ++ i)
    {
        showNumberWithLEDs( i);
        wait( 100000);
    }
}

void main( void)
```

```
{

/*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
PE_ low_ level_ init();
/*** End of Processor Expert internal initialization. ***/


/* Write your code here*/
#pragma warn_ possunwant off


IRQA_ On = IRQA_ GetVal() ? 1 : 0
IRQB_ On = IRQB_ GetVal() ? 1 : 0


for(;;); {
CLEARLEDS;
pattern();


}
#pragma warn_ possunwant reset
}


/* END LEDcontrol */
/*
** ###################################################################
/*
** This file was created by UNIS Processor Expert 03.15 for
** the Freescale DSP56x series of microcontrollers.
**
** ###############################################################
*/
```

5. Bean Wizard Description

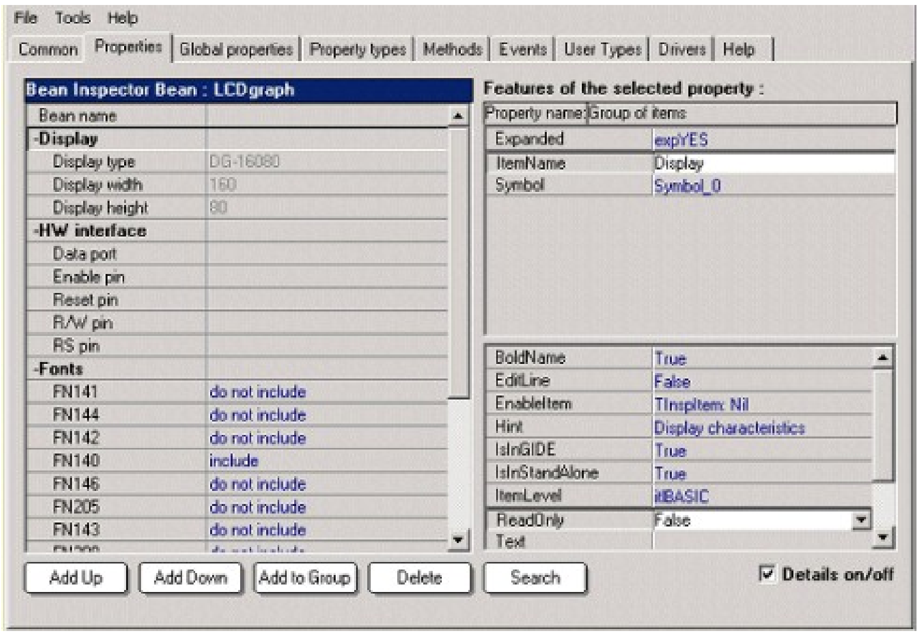
Bean Wizard is a tool dedicated to the creation and edition of Embedded Beans. See chapter *3.2.1 Embedded Beans* for details. It provides a powerful interface for the composition of new beans, and generates the bean files.

Using Bean Wizard, the user can create new beans very quickly and easily without errors in the generated files. The user needs only to determine the Properties, Methods and Events and write the necessary implementations of the methods and events.

Bean Wizard facilitates the reusability of existing Beans and helps edit the source code (quick location, editor, ...).

Bean Wizard is an **external tool**, not a part of Processor Expert.

For further information, see Bean Wizard Help.


File Tools Help
Common | Properties | Global properties | Property types | Methods | Events | User Types | Drivers | Help
Add Up | Add Down | Add to Group | Delete | Search
Details on/off

Bean Inspector Bean : LCDgraph

Bean name	
-Display	
Display type	DG-16080
Display width	160
Display height	80
-HW interface	
Data port	
Enable pin	
Reset pin	
R/W pin	
RS pin	
-Fonts	
FN141	do not include
FN144	do not include
FN142	do not include
FN140	include
FN146	do not include
FN205	do not include
FN143	do not include

Features of the selected property :

Property name: Group of items

Expanded	expYES
ItemName	Display
Symbol	Symbol_0

BoldName	True
EditLine	False
EnableItem	TInspItem: Nil
Hint	Display characteristics
IsInGIDE	True
IsInStandAlone	True
ItemLevel	ilBASIC
ReadOnly	False
Text	

INDEX